MASTER EN SCIENCES INFORMATIQUES

Utilisation des algorithmes flous dans le développement d'un système d'aide à la communication

Uytdenbroek, Marc

Award date:
1995

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Rue Grandgagnage, 21
B-5000 NAMUR

Utilisation des algorithmes flous dans le développement d'un système d'aide à la communication

Marc UYTDENBROEK

Promoteur : Monique Noirhomme

Mémoire présenté en vue
de l'obtention du grade de
Licencié et Maître en informatique

Année académique 1994-1995

Résumé

Le but de ce projet est d'explorer la possibilité d'appliquer la théorie des ensembles flous dans la construction d'un système d'aide à la communication pour les personnes souffrant de problèmes de langage.

Ce travail décrit donc un système d'aide à la communication, appelé FuzzyChat. Il assiste l'utilisateur en lui permettant de changer naturellement et facilement de type de sujets lors de la conversation. FuzzyChat utilise la théorie des ensembles flous pour fournir à l'utilisateur un ensemble de prédictions qui emploient des phrases préstockées lors de conversations. Cette méthode de recherche d'information permet de proposer à l'utilisateur plusieurs phrases, proches au plan du contenu de la phrase courante. De cette manière, le système fournit des prédictions précises pour l'utilisateur, allègeant fortement sa charge cognitive et lui permettant de participer plus naturellement à la conversation.

Abstract

The purpose of this project is to explore the feasibility of applying fuzzy set theory in the design of an augmentative communication system for non-vocal severely physically impaired people.

Then, this paper describes a prototype communication system for non-speakers, FuzzyChat, which has features designed to assist the user to move naturally and easily within and across conversation topics. FuzzyChat uses fuzzy set theory to provide the user with predictive assistance to incorporate prestored text appropriately into conversations. This type of information retrieval method always finds a set number of conversational text items which are close in content to the present text. In this way, the system provides sensible predictions as to the next item to be spoken, and, by taking a great deal of the cognitive burden off the user, allows them to participate more naturally in a conversation.

Remerciements

Je tiens tout d'abord à remercier l'ensemble des personnes du *MicroCentre* (centre de recherche) de l'université de Dundee, en Ecosse. Elles m'ont accueilli et introduit dans leur équipe d'une façon remarquable, facilitant mon adaptation et me permettant de me sentir à l'aise au sein de l'équipe de travail dès le début et pendant mes quatre mois de stage. L'atmosphère de travail du *MicroCentre* fut idéale pour effectuer mes recherches et mon travail.

Dans cette équipe, trois personnes ont été particulièrement précieuses dans la réalisation de ma tâche à Dundee. La première est Marianne Hickey qui, par un suivi hebdomadaire, m'a guidé et conseillé tout au long des quatre mois. Ses suggestions et ses avis m'ont chaque fois fortement aidé dans la réalisation de mon projet. De même, je tiens également à remercier Norman Alm, qui a toujours été présent et disponible pour me soutenir en cas de problème. La collaboration de Nigel Devnani fut, et d'une autre manière, fort appréciée et appréciable. Il a su m'apporter l'expérience de travail qui me manquait dans la réalisation de mon projet. Je tiens également à le remercier pour toute l'aide « extra-professionnelle » qu'il m'a apportée tout au long de mes quatre mois à Dundee.

Je suis aussi reconnaissant à Madame Noirhomme pour deux raisons principales. Je la remercie d'abord pour les nombreux contacts qu'elle a pris avec l'université de Dundee en vue du bon déroulement de mon stage. Elle a été à la base d'une collaboration étroite et, espérons-le durable, entre l'université de Dundee et l'université de Namur. Je la remercie d'autre part pour tout son accompagnement dans la rédaction de ce mémoire.

Enfin, je tiens à remercier mes proches pour le soutien moral et matériel qu'ils m'ont apporté durant l'ensemble de cette année.

Table des matières

Introduction

Dans notre société, un certain nombre d'individus sont atteints de handicaps qui les empêchent de communiquer effectivement avec leurs semblables par la parole. La communication vocale constitue un des plus importants aspects de l'interaction sociale humaine. Elle permet non seulement d'exprimer ses besoins et d'échanger de l'information, mais aussi de créer et de maintenir une relation sociale avec d'autres personnes. La communication est indispensable à chaque individu pour son équilibre vital.

Ces personnes handicapées ressentent souvent un sentiment de frustration et d'isolement vis-à-vis du monde qui les entoure. En effet, elles sont souvent incapables de maîtriser leur environnement et d'exercer une action sur celui-ci. Elles sont, pour la plupart, fort dépendantes d'un petit cercle fermé de proches.

Plusieurs méthodes existent actuellement pour aider ces personnes à interagir avec l'extérieur. L'une d'elles implique l'utilisation d'un ordinateur combiné avec un synthétiseur de voix. Ainsi la personne handicapée se sert de l'ordinateur pour communiquer.

Récemment encore, cette méthode a été employée en utilisant un mécanisme basé sur la production de texte vocalisé, mot par mot ou lettre par lettre. Elle n'a malheureusement donné que peu de résultats positifs, parce que trop lente et trop contraignante.

Actuellement plusieurs chercheurs étudient la possibilité d'utiliser des méthodes basées sur la production de texte vocalisé, phrase par phrase, voire même plus. Cette technique part du principe que les personnes ont généralement tendance à utiliser dans leur conversation les mêmes sortes de phrases et de raconter des histoires semblables à des groupes de personnes différents. Il semble donc sensé d'imaginer des systèmes qui

permettent aux utilisateurs de créer un ensemble d'objets de conversations (les phrases) et de les utiliser quand cela est nécessaire.

Le projet que je vous présente ici se base sur cette méthode, en utilisant un système de prédiction de phrases à l'écran. Il inclut également l'emploi de la théorie des ensembles flous, inventée par Zadeh, dans son système de recherche d'information.

Ainsi, le but de ce projet est d'explorer la possibilité d'appliquer la théorie des ensembles flous dans la construction d'un système d'aide à la communication pour les personnes souffrant de problèmes de langage. Ce projet a été réalisé en collaboration avec le *MicroCentre* de l'université de Dundee en Ecosse et a fait l'objet de plus de 4 mois complets d'analyse de la documentation existante, de recherche, de conception, d'implémentation et de tests.

La rédaction de ce mémoire suit les différentes étapes du travail réalisé à l'université de Dundee et en fournit une description détaillée. La première étape fut une étape de documentation : documentation sur les types de handicaps existants et sur leurs conséquences, documentation sur la théorie des ensembles flous et sur les différents outils utilisés dans le domaine des prothèses de communication. La seconde étape consiste à réaliser le logiciel FuzzyChat.

Ce travail est donc divisé en deux grandes parties. La première, « *Cadre général du projet* », correspond à la phase de documentation et la seconde, « *Création d'un système flou d'aide à la communication : FuzzyChat* », à la phase de conception, d'implémentation et de tests de FuzzyChat.

La première partie regroupe les trois premiers chapitres de ce travail. Le chapitre 1, « *Données de base — Explication du problème* », présente une vision globale des problèmes de communication que peuvent connaître certaines personnes. Nous poursuivons ensuite notre étude par une approche purement théorique du concept des ensembles flous dans le chapitre 2, « *La théorie des ensembles flous* ». Elle nous permet de donner les bases suffisantes pour comprendre l'utilisation de cette théorie dans la seconde partie. Le chapitre 3, « *Les systèmes prédictifs de communication* », nous livre un aperçu des outils existants dans le domaine des prothèses de communication, utilisant un système de prédiction. Il nous montre l'état d'avancement de ces systèmes dans le domaine qui nous concerne.

La seconde partie détaille avec précision le système d'aide à la communication, FuzzyChat. Ainsi le chapitre 4, « *Développement d'un système flou de recherche*

d'information », décrit les différentes fonctions du logiciel, ainsi que son algorithme clé. Nous poursuivons notre recherche par l'analyse et la justification de l'interface construite lors de la création du logiciel, dans le chapitre 5, « *Développement de l'interface du système* ». Le chapitre 6, « *Détails de l'implémentation* », nous apporte quelques détails sur la manière dont le système FuzzyChat a été réalisé techniquement. Nous terminons ce mémoire par l'évaluation du logiciel dans le chapitre 7, « *Evaluation du système et travaux futurs* ». Cette analyse se base notamment sur une enquête réalisée à la fin du projet. Dans ce dernier chapitre, nous évoquons également les différentes possibilités de prolonger ce travail en l'améliorant et en l'intégrant à d'autres structures.

La conclusion de ce mémoire reprend d'une manière synthétique les éléments essentiels de cette recherche, son intérêt, son originalité, ses limites, ainsi que ses possibilités d'approfondissement ultérieur.

Partie I

Cadre général du projet

Chapitre 1

Données de base — Explication du problème

Un peu partout autour de nous, des personnes sont atteintes de problèmes de langage. Même lorsque que ces problèmes sont passagers ou temporaires, ils affectent toujours gravement l'individu atteint. En effet, celui-ci est ainsi coupé de toute possibilité d'exprimer ses besoins, d'échanger de l'information ou d'entretenir une relation personnelle.

Heureusement, de nombreuses méthodes et techniques sont mises en place pour essayer de réduire ces handicaps. L'usage de l'ordinateur peut constituer une solution.

Nous allons d'abord relever ici l'importance de la communication dans la vie de tous les jours. Ensuite nous discuterons des systèmes d'aide à la communication en général, en les définissant et en expliquant leurs buts.

Par la suite, nous parlerons d'un problème particulier aux problèmes de communication, les troubles du langage. On y cherchera également une solution dans les prothèses de communication. Nous finirons ce chapitre par l'étude de la technique de prédiction dans les systèmes de communication.

1.1. Importance de la communication

Pour la plupart d'entre nous, le fait de communiquer est relativement facile et est si naturel que l'on tend à en oublier son importance. Cependant, la capacité de l'être humain à communiquer est aussi vitale que la santé et le développement ou que de se nourrir, de boire ou de respirer [Marlborough, 90].

Nous ne communiquons pas seulement pour faire savoir aux autres nos besoins mais aussi pour échanger de l'information ou exprimer nos émotions. La vie d'une personne privée de l'opportunité d'interagir avec d'autres et d'exercer un contrôle sur son environnement, peut se caractériser par l'isolement ou la frustration.

Le professeur Albert Jacquard[1], à plusieurs reprises, insiste sur l'importance des échanges et de la communication entre personnes humaines :

> « La génétique me fait comprendre pourquoi j'ai un foie, une rate, un cerveau et comment ça marche. Mais elle ne me fait pas comprendre comment 'j'ai' une personne. Par conséquent, il faut que je trouve la source de ma personne ailleurs. Et je la trouve, moi, dans les contacts avec les autres. Par conséquent, la principale richesse qui m'a fait, ce n'est pas tel ou tel gêne mais ce sont les contacts, les échanges que j'ai eus. Construire quelqu'un, c'est lui permettre d'avoir des échanges. Le but d'une société, c'est d'échanger. » [Jacquard, 94a]

> « Profitant de nos malfaçons (...), notre cerveau trop gros, notre larynx trop bas, nous avons imaginé des mises en commun qu'aucun autre animal n'est capable de faire. Nous transmettons des émotions, des projets, des angoisses et avec l'autre, nous créons un lien entre lui et nous. Un lien qui le transforme et qui nous transforme. » [Jacquard, 94b]

En Belgique, il y a approximativement 55 000 personnes qui ne peuvent pas utiliser la parole pour communiquer [Beukelman, 92]. La communication vocale est l'un des plus importants aspects de l'interaction sociale humaine. Cette inaptitude dans la communication est un handicap très contraignant.

---

[1] Polytechnicien, docteur en biologie humaine, diplômé de l'Université de Stanford, membre du Haut-Comité pour le logement des personnes défavorisées, le professeur Albert Jacquard est l'auteur d'une trentaine d'ouvrages.

Voici, pour appuyer l'importance de la communication, le témoignage de Sue Simpson, qui a perdu l'usage de la parole à l'âge de 36 ans :

> « So you can't talk, and it's boring and frustrating and nobody quite understands how bad it really is. If you sit around and think about all the things you used to be able to do, that you can't do now, you'll be a miserable wreck and no one will want to hang around you long. » [Beukelman, 92]

1.2. Système d'aide à la communication

1.2.1. Qu'est-ce qu'un système d'aide à la communication ?

Un système d'aide à la communication est défini dans [Beukelman, 92] comme:

> "(..) an integrated group of components, including the symbols, aids, strategies and techniques used by individuals to enhance communication".

Cette définition insiste sur l'utilisation de plusieurs composants dans un système d'aide à la communication. Dans la définition, le terme *symbols* se réfère à certaines méthodes utilisées dans la représentation tactile, visuelle ou auditive de concepts conventionnels (le braille, la gestuelle, l'ensemble des signes manuels, ...). Il est important de remarquer que l'utilisation de la communication par gestes fait partie intégrante de la définition.

Le terme *aids* est utilisé pour se référer aux outils physiques servant à la transmission ou à la réception de messages. Une *strategy* est une tactique spécifique utilisant les *aids, symbols* et *techniques* pour rendre la communication plus efficace. Et pour finir, le terme *techniques* fait référence à toutes les méthodes de transmission de message (*linear scanning, row-column scanning, encoding, signing, ...*). Ces quatre composants — *symbol, aid, strategy* et *technique* — sont les éléments indispensables que comprend tout système d'aide à la communication.

1.2.2. Buts d'un système d'aide à la communication

Le but d'un système d'aide à la communication est de rendre capable un individu d'agir dans une série d'interactions. Une étude[1] a identifié 4 buts que doit remplir tout système d'aide à la communication : 1) Expression des besoins; 2) Transfert d'information; 3) Relations sociales et 4) Etiquette sociale. Les caractéristiques de ces types d'interactions sont résumées dans le Tableau 1.1.

L'*Expression des besoins* permet à un individu de faire savoir à une autre personne la nature de ses besoins (par exemple, une personne faisant une commande dans un restaurant). Ici, le contenu du message est fort important, le vocabulaire est relativement prédictible et le taux de rapidité d'expression du message est crucial. C'est probablement (selon [Beukelman, 92]) ce haut degré de prédictibilité qui explique pourquoi le vocabulaire des *besoins* tend souvent à prédominer dans beaucoup de systèmes d'aide à la communication.

Le *Transfert d'information* est le plus difficile besoin à rencontrer car son but est de partager de l'information plutôt que d'essayer de réguler des attitudes. On peut prendre comme exemple un enfant racontant à son professeur ce qu'il a fait pendant son dernier week-end ou un adulte répondant à des questions durant une entrevue pour un emploi. Comme dans le cas de l'*Expression des besoins*, le contenu du message est important. Le vocabulaire est relativement nouveau (et donc non-prédictible), ce qui permet à l'utilisateur de s'exprimer sur des sujets fort différents. Le taux de rapidité d'expression du message reste encore d'une importance extrême.

La communication liée à la *Relation sociale* diffère des deux premiers types. Le but de ce type d'interaction est d'établir, de maintenir et de développer une relation sociale personnelle. De ce fait, le contenu de ce message est moins important que l'interaction elle-même et n'est habituellement pas prédictible. On peut prendre, comme exemple, un enfant racontant une blague à son camarade ou une femme exprimant ses sentiments de sympathie à un ami dont la mère est morte récemment. Dans ce type d'interaction, le taux d'émission du message et son contenu sont secondaires par rapport au sentiment d'établir une relation.

---

[1] Light J. (1988). Interaction involving individuals using AAC systems: State of the art and future directions. *Augmentative and Alternative Communication*, 4, 76.

Caractéristiques	*Expression des besoins*	*Transfert d'information*	*Relations sociales*	*Etiquette sociale*
But de l'interaction	Exprimer un besoin.	Partager l'information.	Etablir, maintenir et développer une relation personnelle.	Se conformer aux conventions sociales.
Objet de l'interaction.	Actions ou objets désirés.	Information.	Relation interpersonnelle.	Convention sociale.
Durée de l'interaction	Limitée	Peut être longue	Peut être longue	Limitée
Contenu de la communication	Important	Important	Pas important	Pas important
Prédictibilité de la communication	Fortement prédictible.	Pas prédictible	Peut être prédictible.	Fortement prédictible
Etendue de la communication	Etendue limitée	Large étendue	Large étendue	Etendue très limitée
Vitesse de la communication	Importante	Importante	Peut ne pas être importante	Importante
Tolérance pour les interruptions dans la communication.	Peu de tolérance	Peu de tolérance	Plus de tolérance	Peu de tolérance
Nombre de participants	Habituellement deux.	Deux, petits ou grands groupes.	Habituellement deux ou de petits groupes	Deux, petits ou grands groupes.
Importance du communicateur	Important	Important	Pas important	Important.
Partenaire	Famille ou autres	Famille ou autres	Habituellement famille	Famille ou autres.

Tableau 1.1: Caractéristiques d'une interaction à travers ses buts sociaux

Le but du quatrième type d'interaction, l'*Etiquette sociale,* est de se conformer aux conventions de politesse durant une interaction avec une autre personne. La communication est souvent brève et contient des expressions prédictibles. Un enfant disant « s'il vous plaît » ou « merci » à sa grand mère ou un adulte répondant de façon cohérente à un commentaire du temps en sont deux exemples. On peut remarquer que ces messages ressemblent étroitement aux messages de l'*Expression des besoins*, parce que le taux d'émission du message et la précision du message sont deux facteurs importants pour le succès de l'interaction.

1.2.3. Les outils d'aide à la communication

Dans les dernières années, un nombre considérable d'outils d'aide à la communication est apparu sur le marché. En particulier, les nouvelles technologies ont permis d'augmenter la communication des personnes handicapées d'une façon telle qu'on n'aurait pas pu l'imaginer il y a quelques années. Les traitements de texte permettent à ces personnes d'écrire sans devoir manipuler du papier. Les ordinateurs rendent l'accès possible à des sources d'information pour ceux qui ne savent pas manipuler les livres. De nouveaux développements en télécommunication permettent aux personnes *moins valides* de communiquer sur des grandes distances.

Cependant, les outils d'aide à la communication ne doivent pas nécessairement incorporer les dernières technologies. De nombreux outils simples peuvent être utiles pour assister la communication.

Néanmoins, une aide efficace dépend de beaucoup de facteurs. Il faut attacher une grande importance à l'évaluation des besoins du futur utilisateur. Est-ce que l'aide est appropriée à la situation de la personne qui l'utilise ? Est-ce que l'outil d'aide à la communication est fiable et facile à utiliser ? Le plus important n'est-il pas que la personne soit motivée pour l'utiliser ? Beaucoup d'aides restent inutilisées ou sous-utilisées parce qu'elles ne répondent pas aux besoins de l'utilisateur.

Voici quelques exemples d'outils d'aide à la communication (plus de détails et des informations complémentaires peuvent se trouver dans [Marlborough, 90]) :

- BLISSYMBOLICS COMMUNICATION SYSTEM : The Blissymbolics system est maintenant utilisé dans beaucoup de pays au moyen d'un langage symbolique. Les symboles sont basés sur leur concept et leur signification plutôt que sur leur son. Certains symboles sont des pictogrammes désignant les objets qu'ils représentent. D'autres symboles sont des idéogrammes représentant plutôt des idées.
- The Speech Synthesisers : Les synthétiseurs de voix peuvent être utilisés en conjonction avec un ordinateur pour aider les personnes souffrant de problème de langage.
- CHAILEY HEADPOINTER : Le pointeur de tête (*headpointer*) permet d'utiliser de façon ordinaire le clavier d'une machine à écrire ou d'un ordinateur. Il permet aussi de dessiner ou de tourner les pages.

1.3. Les problèmes de langage

1.3.1. Qui sont ces personnes ?

Les personnes souffrant de problèmes de langage sont des personnes de toutes les catégories d'âge, de tout milieu social et de toute culture. La caractéristique qu'elles ont en commun est qu'elles requièrent toutes une assistance pour parler. Une définition plus exacte nous est donnée par *the American Speech-Language-Hearing Association* :

> « Individuals with severe communication disorders are those for whom gestural, speech, and/or written communication is temporarily or permanently inadequate to meet all of their communication needs. For these individuals, hearing impairment is not the primary cause for the communication impairment. Although some of these individuals may be able to produce a limited amount of speech, it is inadequate to meet their varied communication needs. Numerous terms that were initially used in the field, but are now rarely mentioned, include speechless, nonoral, nonvocal, nonverbal and aphonic. » [Beukelman, 92]

Différentes maladies sont la cause de ces problèmes de langage. On peut classer ces maladies en trois grandes catégories[1] :

- **Problèmes d'articulation :**

  - Anarthrie : Incapacité d'articuler les mots à la suite d'une lésion cérébrale.

  - Dysarthrie : Difficulté à articuler les mots, due à une paralysie ou à une ataxie[2] des centres nerveux.

  - Dyspraxie orale : Difficulté motrice d'un sujet à articuler, liée à un trouble de la représentation corporelle et de l'organisation spatiale.

- **Problèmes de voix :**

  - Aphonie : Extinction de la voix.

  - Dysphonie : Modification pathologique du timbre de la voix (voix cassée, rauque, éteinte).

---

[1] Les définitions sont retirées du Petit Larousse illustré, 1993.

[2] Absence de coordination des mouvements, caractéristique de certaines maladies neurologiques.

- **Problèmes de langage :**

  - Aphasie : Perte de la parole ou de la compréhension du langage à la suite d'une lésion du cortex cérébral.

  - Dysphasie : Retard important du langage de l'enfant.

1.3.2. Les prothèses de communication — Utilisation de l'ordinateur

Heureusement, il existe beaucoup de méthodes pour aider ces personnes à communiquer. Ces méthodes sont appelées *prothèses de* communication. Ce sont des outils, au sens large, qui permettent de remplacer ou d'accentuer l'usage de la parole.

Avant de détailler les différents types de prothèses de communication existant, il est important de faire une remarque sur l'utilisation de celles-ci. Il faut bien considérer le point suivant : Ces personnes sont généralement toujours entourées des mêmes gens (en règle générale, la famille) et la relation qui existe entre eux est très forte. Les proches de la personne handicapée et elle-même ont, au fur et à mesure, généralement créé une relation de complicité, où le moindre geste a une signification.

Il peut être dès lors inutile d'introduire une prothèse de communication au sein d'un milieu tel que celui-là, puisque les buts, (voir *1.2.2. Buts d'un système d'aide à la communication*) — « l'expression des besoins », « les relations sociales » et « l'étiquette sociale » — sont atteints par des moyens contournés. Seul le but « Transfert d'information » ne serait pas encore atteint.

De plus, il ne faut pas négliger les problèmes liés à l'introduction d'une prothèse de communication. En effet, certaines prothèses peuvent complètement bouleverser le mode de vie de la personne concernée, changeant les relations de complicité qui s'étaient établies.

L'introduction d'une prothèse de communication ne doit pas être prise à la légère et mérite d'être étudiée en profondeur.

Un des premiers types de prothèse de communication est l'utilisation de tableaux de lettres, de mots ou de phrases, surlesquels un choix peut être effectué. L'utilisateur désigne une lettre, un mot ou une phrase pour faire part à son conversant ce qu'il désire communiquer. Il est évident que ce mode de communication est fastidieux et extrêmement lent.

Avec l'apparition, de plus en plus grande, des PC sur le marché, une autre méthode consiste dans l'utilisation de l'ordinateur. Les personnes ayant des difficultés à s'exprimer utilisent des systèmes de communication — des logiciels — qui créent du son, via un synthétiseur de voix.

Il reste vrai, cependant, que même les derniers systèmes disponibles n'atteignent pas la vitesse d'une conversation normale. Actuellement, les ordinateurs basés sur un système de communication opèrent lettre par lettre ou mot par mot, ce qui est très lent. Une conversation sans entrave peut aller de 150 à 250 mots à la minute, alors que dans les systèmes de communication actuels, le taux de mots émis par minute ne dépassent que rarement les 10 mots [Beukelman, 92].

L'idée de stocker des messages entiers plutôt que des lettres et des mots est ensuite apparue. En effet, beaucoup de messages, utilisés quotidiennement, sont réutilisables (exemples : commentaires sur le temps, blagues, histoires de vacances, etc.). En sélectionnant des *messages* entiers plutôt que des mots, la vitesse de communication peut s'accroître de façon prodigieuse.

Cependant, les systèmes commerciaux actuels demandent aux utilisateurs d'assigner un code d'accès unique à chaque nouveau *message*. L'utilisateur doit alors se souvenir de ce code d'accès unique lorsqu'il désire faire appel à un message particulier dans une conversation. Quand un système stocke des centaines ou même des milliers de messages, cela devient difficile, voire même impossible, pour l'utilisateur de se souvenir de ces codes. En pratique, les personnes ne peuvent s'exprimer elles-mêmes avec de tels systèmes parce qu'il y a une restriction sévère dans le nombre des différents messages qu'elles peuvent *parler*. Le résultat est que cela provoque à nouveau chez l'utilisateur une frustration dans sa capacité à prendre part pleinement au monde qui l'entoure.

D'autres techniques ont aussi été mises en oeuvre, dans le domaine de l'*information retrieval*, pour ne pas devoir faire appel à ces codes d'accès uniques. L'usage de la technique de prédiction est aussi un domaine exploitable.

1.4. Utilisation de la technique de prédiction

1.4.1. Introduction

Il y a un nombre important de stratégies pour accélérer la vitesse de communication et augmenter le *timing* des messages. Une de ces stratégies est la stratégie de prédiction dans les systèmes de communication. La prédiction de messages peut être définie comme suit :

> « Message prediction is a dynamic retrieval process in which options offered to the user change based on the portion of the message that has already been formulated » [Beukelman, 92].

Les algorithmes de prédiction sont généralement séparés en 3 catégories :

- **Single Letter Prediction** : Les langages orthographiques sont souvent organisés de telle manière que lorsqu'on frappe une lettre, on peut tenter de prédire la suivante. Prenons un exemple dans la langue française : il y a une forte probabilité qu'après la lettre *q* vienne la lettre *u*. De même certaines combinaisons — telles que *tion, que, elle, eur, ...* — apparaissent plus souvent que d'autres. Le principe est de construire un système de prédiction automatique basé sur la probabilité d'apparition de ces lettres. Ainsi lorsqu'une lettre est sélectionnée, le système en propose une série d'autres qui ont la plus grande probabilité d'être sélectionnées.
- **Word Level Prediction** : Certains systèmes ont inclu des algorithmes qui contiennent des informations sur l'organisation syntaxique du langage. Les prédictions offertes sont basées sur des règles grammaticales.
- **Phrase/Sentence Level Prediction** : Une équipe de chercheurs, au MicroCentre de l'Université de Dundee (Ecosse), a développé plusieurs logiciels de communication qui incorporent des algorithmes sophistiqués de prédiction d'unité de langage plus grande que le simple mot. Par exemple, CHAT [Newell, 92] inclut des segments de conversation et TALKBAC [Waller, 93] utilise la technique de prédiction de phrases.

1.4.2. Sentence Level Prediction — Fuzzy logic theory

En plus de ces techniques, d'autres projets utilisant des systèmes de prédiction de phrases ont été développés. Ces systèmes ont établi des liens entre les messages (phrases) en se basant sur une *fuzzy information retrieval method*. Ce type de recherche d'information permet de retrouver un ensemble de messages (dans la base de données) qui sont proches en contenu du message courant. De cette manière, le système fournit une prédiction de messages pour l'utilisateur. Ainsi, en mettant plus *d'intelligence* dans le système, on laisse l'utilisateur interagir librement.

Le but de ce projet est d'explorer la possibilité d'appliquer la théorie des ensembles flous dans la construction d'un système d'aide à la communication pour les personnes souffrant de problèmes de langage.

Chapitre 2

La théorie des ensembles flous

Dans ce chapitre, nous allons passer rapidement en revue les différents éléments de la théorie des ensembles flous. Notre but ici n'est pas de donner une vue complète et exhaustive de la théorie des ensembles flous mais d'en fournir les éléments essentiels en vue de la compréhension des chapitres suivants.

Après une petite introduction théorique, nous allons définir ce qu'est un ensemble flou. Nous étudierons ensuite les opérateurs et les transformateurs que l'on peut appliquer sur ces ensembles. Nous finirons ce chapitre en illustrant le principe du raisonnement flou.

2.1. Introduction

2.1.1. Lofti Zadeh

Qui peut mieux décrire une théorie que son inventeur ? Voici donc quelques mots de Lofti A. ZADEH introduisant la théorie des ensembles flous (préface de [Cox, 94]) :

« My 1965 paper [Zadeh, 65] on fuzzy sets was motivated in large measure by the conviction that traditional methods of systems analysis are unsuited for dealing with systems in which relations between variables do not lend themselves to representation in terms of differential or difference equations. Such systems are the norm in biology, sociology, economics and, more generally, in fields in which the systems are humanistic rather than mechanistic in nature.

Traditional methods of analysis are oriented toward the use of numerical techniques. By contrast, much of human reasoning involves the use of variables whose values are fuzzy sets. This observation is the basis for the concept of a linguistic variable, that is, a variable whose values are words rather than numbers. The concept of a linguistic variable has played and is continuing to play a key role in the applications of fuzzy set theory and fuzzy logic. »

Lotfi A. Zadeh poursuit ensuite dans sa préface en insistant sur l'importance de l'utilisation de variables linguistiques :

« The use of linguistic variables represents a significant paradigm shift in systems analysis. More specifically, in the linguistic approach the focus of attention in the representation of dependencies shifts from difference and differential equations to fuzzy if-then rules in the form *if X is A then Y is B*, where X and Y are linguistic variables and A and B are their linguistic values, e.g. *if Pressure is high then Volume is low*. Such rules serve to characterise imprecise dependencies and constitute the point of departure for the construction of what might be called the calculus of fuzzy rules. »

Zadeh insiste aussi sur le fait que beaucoup de problèmes proches du monde réel sont loin d'être simples à modéliser. La théorie des ensembles flous ne remplace pas les systèmes déjà existants mais fournit un outil supplémentaire pour trouver des solutions de modélisation à ces problèmes.

Dans leur article [Gupta, 79], Rammohan K. Ragade et Madan M. Gupta prolongent la pensée de Zadeh. Ils disent que le cerveau humain possède certaines caractéristiques

spéciales qui le rendent capable de comprendre et de raisonner dans un environnement flou ou imprécis. Le cerveau arrive à fonder une décision sur base de phénomènes vagues. Le développement d'outils comprenant des concepts flous est un pas de plus en direction du traitement des problèmes humains.

2.1.2. Origine de la théorie des ensembles flous

Comme le conçoit L. Zadeh, son inventeur, la théorie des ensembles flous fournit une méthode qui permet de réduire la complexité des systèmes. Au départ, Zadeh fut préoccupé par le rapide déclin de l'information dans les modèles mathématiques traditionnels lorsque la complexité du système augmentait.

Il réalisa aussi qu'une part importante de cette complexité provenait de la façon dont les variables étaient représentées et manipulées. Quand ces variables ne peuvent que représenter des phénomènes qui existent ou qui n'existent pas, la mathématique nécessaire à évaluer des opérations de limite devient extrêmement complexe. Zadeh précise cela dans son principe d'incompatibilité:

> « [A]s the complexity of a system increases, our ability to make precise and yet significant statements about its behaviour diminishes until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive characteristics. » [Zadeh, 70]

Dans cette vue, les mécanismes sous-jacents sont représentés linguistiquement plutôt que mathématiquement. Zadeh est convaincu que l'être humain ne raisonne pas en termes de symboles et de nombres discrets mais en termes d'ensembles flous. La transition d'une catégorie à une autre est graduelle. C'est à partir de cette idée et en se basant sur les travaux de Max Black et de Jan Lukasiewicz que Zadeh proposa le concept d'ensemble flou.

2.2. Les ensembles flous

2.2.1. Introduction

La majorité des phénomènes que nous rencontrons dans la vie de tous les jours sont imprécis, c'est-à-dire qu'ils ne possèdent pas une définition exacte de leur nature. Cette imprécision peut être associée à leur forme, à leur position, à leur couleur, à leur texture

ou même à leur sémantique. Dans beaucoup de cas, le même concept peut avoir différents degrés d'imprécision selon le contexte ou le temps. Un jour chaud en hiver ne signifie pas la même chose qu'un jour chaud en été. L'exacte définition du passage de tiède à chaud est également imprécise. Ce type d'imprécision, associé au phénomène continu, peut s'appliquer dans toutes les matières : sociologie, physique, biologie, finance, marketing, psychologie, etc.

Un ensemble flou est une fonction qui fait correspondre une valeur à un nombre compris entre 0 et 1. Ce nombre indique le degré d'appartenance de la valeur examinée à l'ensemble flou. Un degré d'appartenance de 0 signifie que la valeur n'appartient en aucune manière à l'ensemble. Un degré de 1 signifie que la valeur examinée est pleinement représentative de l'ensemble. Comme simple exemple, considérons l'idée d'un LONG projet. La Figure 2.1 illustre comment ce concept peut être représenté.

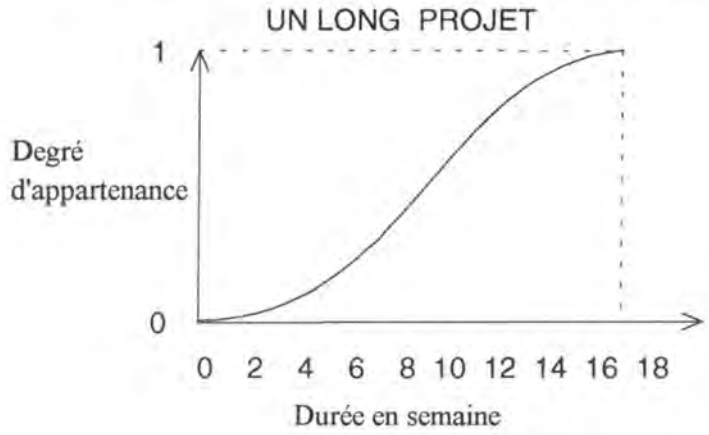

Figure 2.1 : Idée d'un LONG projet

Les membres de cet ensemble sont des durées, en semaines, pour un projet. L'ensemble flou indique à quel degré une durée est considérée comme un LONG projet. Au fur et à mesure que le nombre de semaines augmente, notre croyance que le projet est LONG augmente. Un projet de deux semaines n'est pas considéré comme un LONG projet; un projet de 10 semaines est considéré modérément comme un LONG projet et un projet de 15 semaines ou plus est définitivement un LONG projet. Bien sûr, l'actuelle définition de ce qui est un LONG projet dépend du contexte dans lequel il est utilisé. Pour certains modèles, un projet de 2 ou 3 semaines seulement peut être déjà considéré comme un long projet.

2.2.2. Représenter l'imprécision avec les ensembles flous

Les ensembles classiques (bien définis)

Prenons l'ensemble (être) GRAND pour le sexe masculin. La question à se poser est « A partir de quelle valeur considère-t-on comme grande la taille d'un homme ? ». Dans la théorie classique des ensembles, nous sommes forcés de choisir un point arbitraire, soit par exemple 1,80 mètre. La limite entre ce qui est dans l'ensemble et ce qui ne l'est pas est bien précise. La fonction caractéristique de l'ensemble peut être décrite comme ceci :

$$\mu_{GRAND} = \{Hauteur \geq 180cm\}$$

Donc, toute personne égale ou plus grande à 180 centimètres est considérée comme grande et appartient à l'ensemble. Le graphe d'appartenance de cet ensemble est décrit à la Figure 2.2.

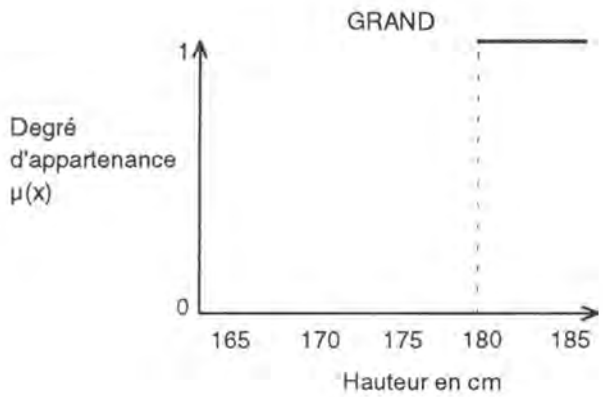

Figure 2.2 : Le concept GRAND dans la théorie classique des ensembles

La fonction caractéristique de cet ensemble révèle sa nature booléenne. En effet, dans cet exemple, les valeurs du domaine (les hauteurs) de l'ensemble GRAND, soit n'appartiennent pas du tout à l'ensemble (cas des valeurs < 180 cm), soit appartiennent complètement à l'ensemble (cas des valeurs ≥ 180 cm). Il n'y a pas d'étape intermédiaire entre les deux. Tous les ensembles classiques possèdent ce type de fonction caractéristique.

Les ensembles flous

L'idée de GRAND, illustrée à la Figure 2.3, est l'exemple classique des ensembles flous et montre les propriétés intrinsèques de l'espace flou. Le domaine de cet ensemble, se trouvant le long de l'axe horizontal, s'échelonne de 160 à 190 cm. Le degré

d'appartenance à cet ensemble, se trouvant le long de l'axe vertical, varie de 0 (pas d'appartenance) à 1 (appartenance complète). Il est connecté avec son domaine, dans ce cas, par une simple courbe linéaire (la hauteur étant directement proportionnelle à la grandeur).

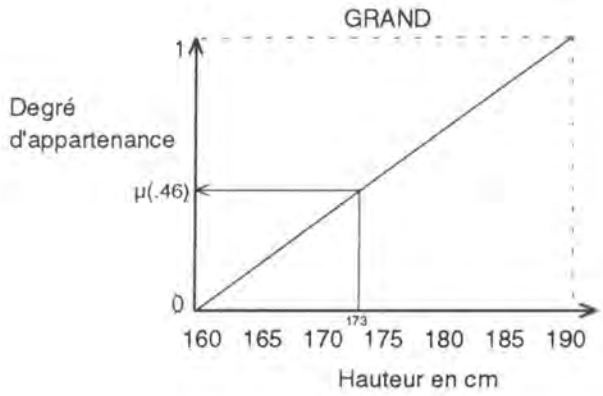

Figure 2.3 : Le concept GRAND dans la théorie des ensembles flous

Prenons maintenant, une valeur dans le domaine de cet ensemble et calculons son degré d'appartenance : 173 cm de hauteur a un degré d'appartenance de [0.46] à l'ensemble GRAND.

Si la valeur (pour la hauteur) vaut moins que 160 cm, son degré d'appartenance sera égal à 0. Si la valeur (pour la hauteur) vaut 190 cm ou plus, son degré d'appartenance à l'ensemble sera alors de 1. Cette fonction d'appartenance, comme nous pouvons le voir, est interprétée comme une mesure de compatibilité entre une valeur du domaine et la courbe de cet ensemble.

Dans le cas de 173 cm, 0.46 d'appartenance signifie qu'il (une personne de 173 cm de hauteur) est modérément compatible avec l'idée de *grandeur* représentée par l'ensemble GRAND. Lorsque son degré d'appartenance diminue vers 0, l'exemple devient de moins en moins compatible avec la sémantique que veut exprimer l'ensemble GRAND. Lorsque son degré augmente vers 1, l'exemple devient de plus en plus compatible avec les propriétés sémantiques de l'ensemble flou.

Comme le montre la Figure 2.4, un ensemble flou se compose de trois éléments :

- **L'axe horizontal** composé d'une série de nombres réels augmentant de façon monotone, qui constitue la population de l'ensemble flou (le domaine).

- **L'axe vertical** allant de 0 à 1 et représentant le degré d'appartenance à l'ensemble flou.
- **La surface de l'ensemble flou** elle-même connectant un élément du domaine avec son degré d'appartenance dans l'ensemble.

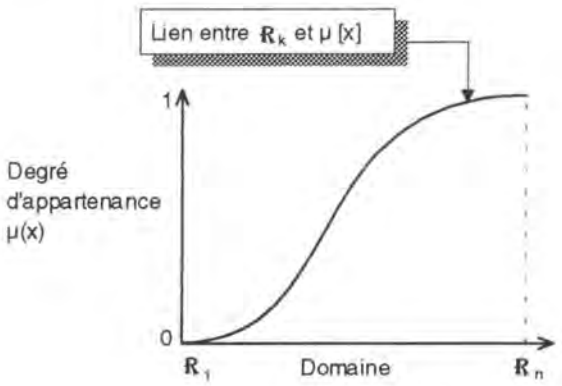

Figure 2.4 : Structure générale d'un ensemble flou

Ce degré d'appartenance peut aussi être vu comme la fonction de vérité de l'ensemble flou, car il établit la correspondance entre un élément du domaine et sa valeur de vérité, indiquant le degré d'appartenance de l'élément à l'ensemble.

$$\mu_A(x) \leftarrow f(x \in A)$$

Equation 2.1: Fonction générant la vérité floue

La *fonction générant la vérité floue* (Equation 2.1) n'est pas symétrique. Chaque point du domaine a un degré d'appartenance, mais on peut associer un degré d'appartenance à plusieurs points du domaine (pour les ensembles flous convexes).

Les ensembles flous encodent l'imprécision à travers leur surface. En fait, la forme de la courbe représente sémantiquement le concept que l'on veut représenter.

2.2.3. Quelques exemples

Dans cette partie, nous allons brièvement exposer plusieurs types de courbes d'ensembles flous. Le premier type, le plus simple, est l'ensemble flou représenté par une surface proportionnellement linéaire. Cette surface peut être croissante ou décroissante (Figure 2.5).

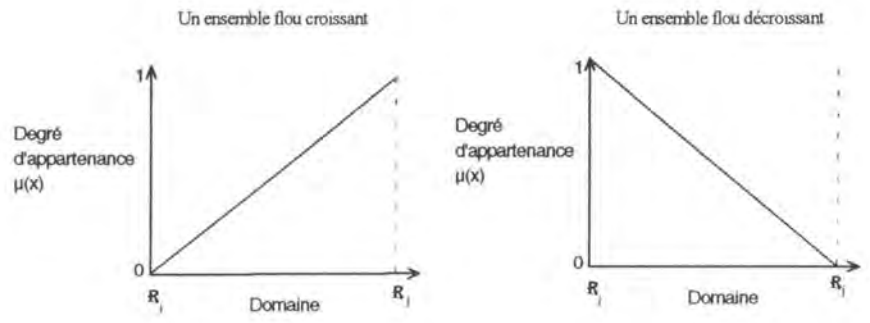

Figure 2.5 : Ensembles flous linéaires croissants et décroissants

De la même manière, un ensemble flou peut être représenté par une surface non linéaire croissante ou décroissante (Figure 2.6). Les surfaces sont aussi appelées des *sigmoïdes* ou des *S-courbes*.

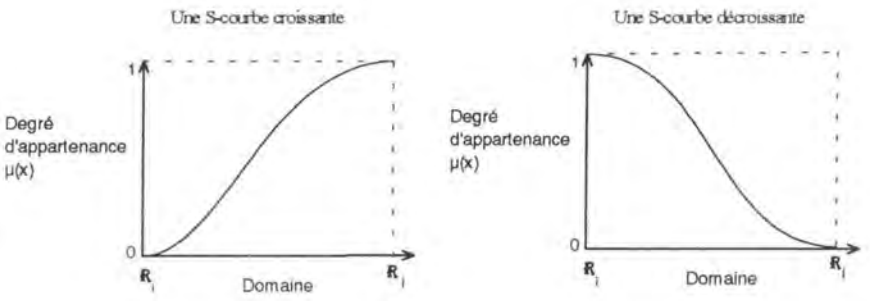

Figure 2.6 : Ensembles flous non linéaires croissants et décroissants

C'est grâce à ce type de courbe que l'on peut représenter les proportions dans la théorie des ensembles flous. Ainsi les ensembles flous GENERALEMENT et LA PLUPART peuvent être conceptualisés (Figure 2.7).

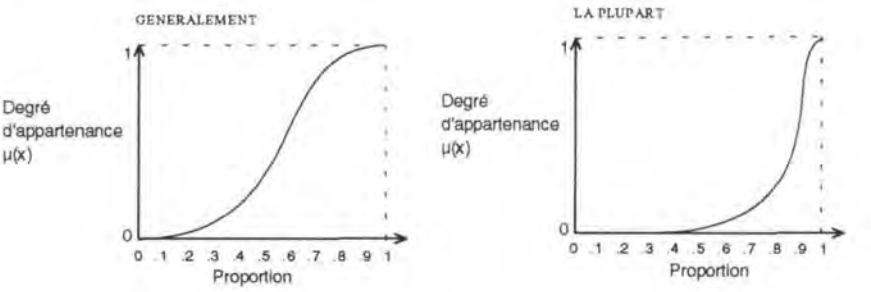

Figure 2.7 : Deux exemples flous représentant des proportions

Une autre classe importante d'ensembles flous est l'approximation d'une valeur centrale. Ce sont les courbes dites *en cloche* (Figure 2.8)

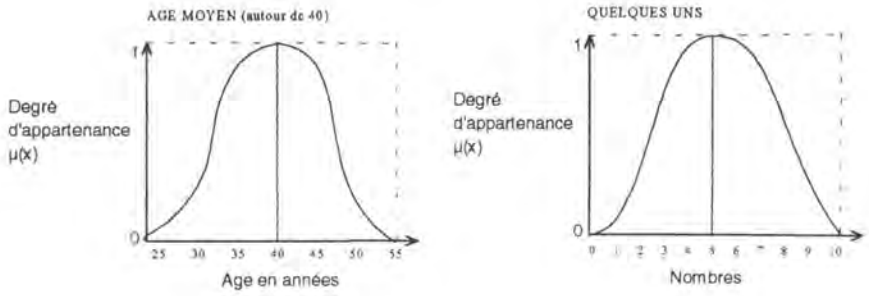

Figure 2.8 *:* Deux exemples flous représentant des approximations

Il existe également des ensembles flous représentés triangulairement. Certains ensembles peuvent avoir aussi des courbes irrégulières ou possédant des points arbitraires.

Pour terminer, voici un exemple dans lequel il existe plusieurs ensembles flous définis sur un même domaine (Figure 2.9). En effet, chaque triangle correspond à un ensemble flou (TRES FROID, FROID, NORMAL, CHAUD, TRES CHAUD).

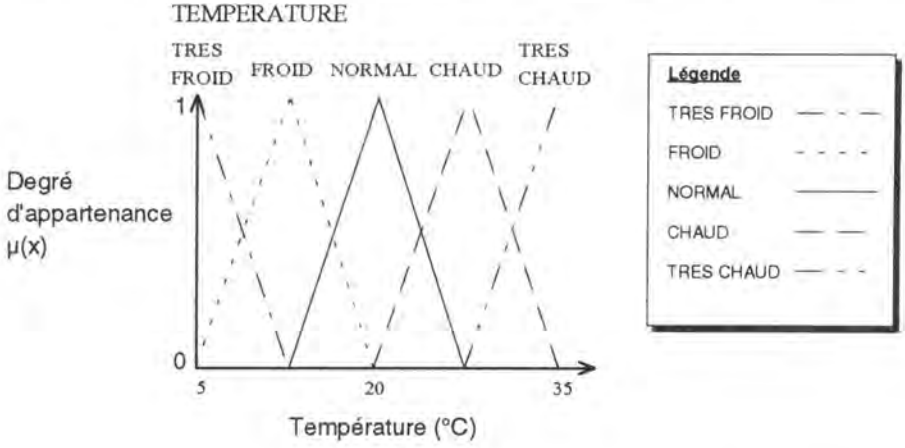

Figure 2.9 : Plusieurs ensembles flous définis sur le même domaine

2.3. Les opérateurs flous

Ayant examiné la structure des ensembles flous, nous allons voir à présent les différentes opérations que l'on peut appliquer sur ces ensembles. Ces opérateurs sont l'intersection et l'union de deux ensembles et le complément d'un ensemble flou.

Nous allons nous en tenir aux opérateurs définis initialement par Zadeh, tout en sachant qu'il en existe d'autres, plus complexes et moins restrictifs. Ceux-ci sont utilisés dans des cas bien précis que nous ne verrons pas ici (le lecteur intéressé pourra consulter [Cox, 94]).

2.3.1. L'intersection de deux ensembles flous

La définition de l'intersection de deux ensembles flous, proposée par Zadeh est la suivante :

Soit A et B, deux ensembles flous:

$$A \cap B = \min\left(\mu_A[x], \mu_B[y]\right)$$

où x et y représentent des points du domaine de A et de B respectivement.

Définition 2.1 : L'intersection de deux ensembles flous

Prenons un exemple concret pour illustrer cela. Supposons deux ensembles flous : l'AGE MOYEN et (être) GRAND (Figure 2.10).

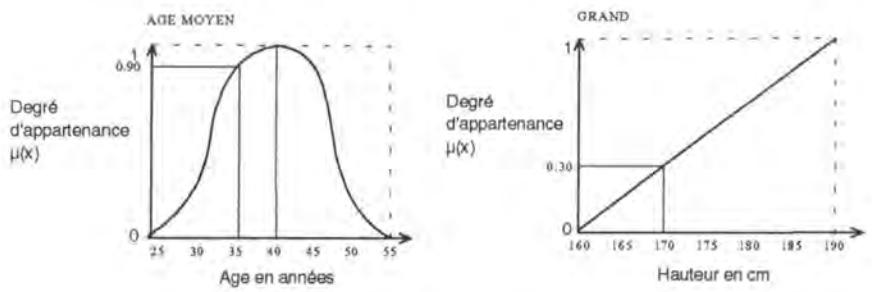

Figure 2.10 : Deux ensembles flous : AGE MOYEN et GRAND

Supposons également une personne de 35 ans mesurant 170 cm. Son degré d'appartenance à l'ensemble AGE MOYEN est de 90 % et à l'ensemble GRAND est de 30 %.

L'intersection de ces deux ensembles (pour cette personne) nous sera donnée en prenant le minimum de ces deux valeurs :

$$AGE\_MOYEN \cap GRAND = \min(\mu_{AGE\_MOYEN}[35], \mu_{GRAND}[170])$$
$$AGE\_MOYEN \cap GRAND = \min(0.9, 0.3) = 0.3 = 30\%$$

La représentation de l'espace flou d'intersection peut être vue en superposant les deux ensembles flous et en prenant leur minimum (Figure 2.11).

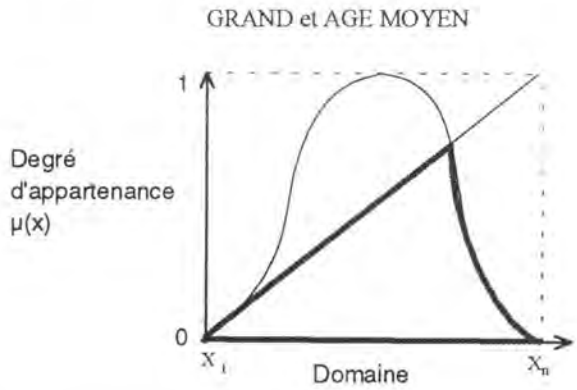

Figure 2.11 : Intersection de deux ensembles flous

2.3.2. L'union de deux ensembles flous

La définition de l'union de deux ensembles flous, proposée par Zadeh est la suivante :

Soit A et B, deux ensembles flous :

$$A \cup B = \max(\mu_A[x], \mu_B[y])$$

où x et y représentent des points du domaine de A et de B respectivement.

Définition 2.2 : L'union de deux ensembles flous

Reprenons les mêmes ensembles flous que précédemment (Figure 2.10) et la même personne (35 ans et 170 cm). L'union de ces deux ensembles (pour cette personne) nous sera donnée en prenant le maximum de leur degré d'appartenance :

$$AGE\_MOYEN \cup GRAND = \max(\mu_{AGE\_MOYEN}[35], \mu_{GRAND}[170])$$
$$AGE\_MOYEN \cup GRAND = \max(0.9, 0.3) = 0.9 = 90\%$$

La représentation de l'espace flou d'union peut être vue en superposant les deux ensembles flous et en prenant leur maximum (Figure 2.12).

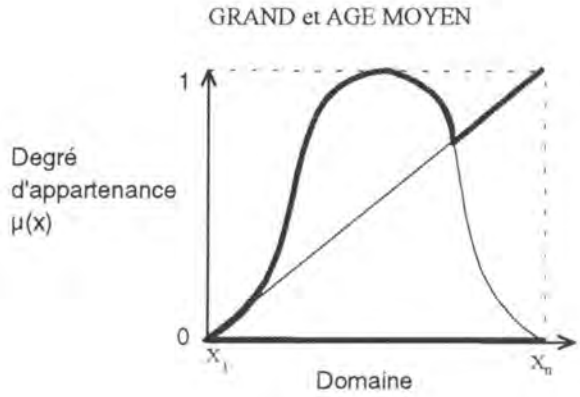

Figure 2.12 : L'union de deux ensembles flous

2.3.3. Le complément d'un ensemble flou

La définition du complément d'un ensemble flou, proposée par Zadeh est la suivante :

Soit A, un ensemble flou :

$$\neg A = 1 - \mu_A[x]$$

où x représente un point du domaine de A.

Définition 2.3 : Le complément d'un ensemble flou

Pour reprendre encore une fois notre exemple (Figure 2.10), le complément de l'ensemble AGE MOYEN pour la personne de 35 ans est de 0.10 (1 - 0.90). De même le complément de l'ensemble GRAND pour la personne de 170 cm est de 0.70 (1 - 0.30).

La Figure 2.13 nous donne une représentation graphique du complément de ces deux ensembles.

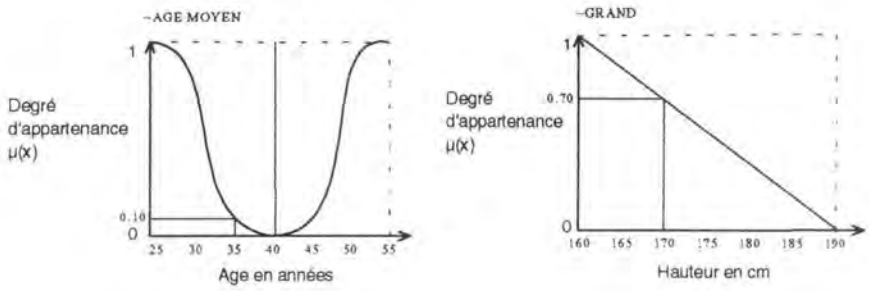

Figure 2.13 : Complément des ensembles AGE MOYEN et GRAND

2.4. Les transformateurs de courbes (hedges)

Nous allons, dans cette partie, rapidement étudier les transformateurs (*hedges*) de courbes d'ensembles flous. Un *hedge* modifie la courbe d'un ensemble flou et change sa fonction d'appartenance.

2.4.1. Types de transformateurs

Un premier type de transformateur : les *hedges* qui intensifient (représentés par *très*, *extrêmement*, etc.) les caractéristiques des ensembles flous ou qui les atténuent (représentés par *plutôt*, *assez*, *passablement*, etc.).

L'effet de ces transformateurs a pour but de réduire (pour les intensificateurs) ou d'élargir (pour les atténuateurs) l'espace de la région floue. Ainsi, pour l'ensemble GRAND et pour le transformateur *très*, la caractéristique suivante est de mise :

$$\mu_{GRAND}[x] \geq \mu_{très\_GRAND}[x]$$

La Figure 2.14 nous montre l'ensemble flou GRAND, *très* GRAND et *extrêmement* GRAND.

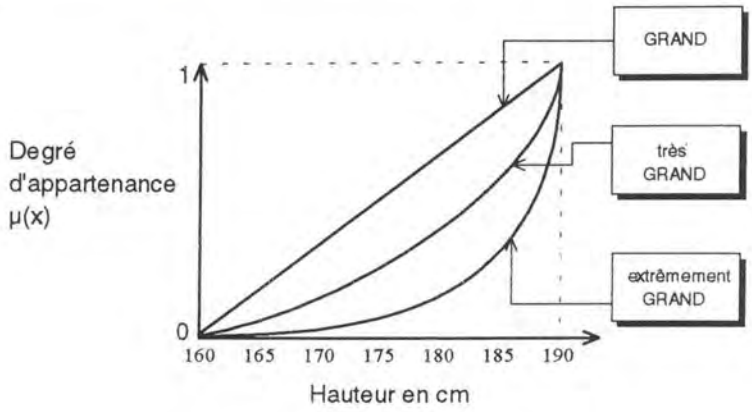

Figure 2.14 :
Transformateurs (très, extrêmement) qui intensifient la courbe de l'ensemble flou

De même, l'ensemble flou AGE MOYEN peut être atténué par un transformateur (Figure 2.15).

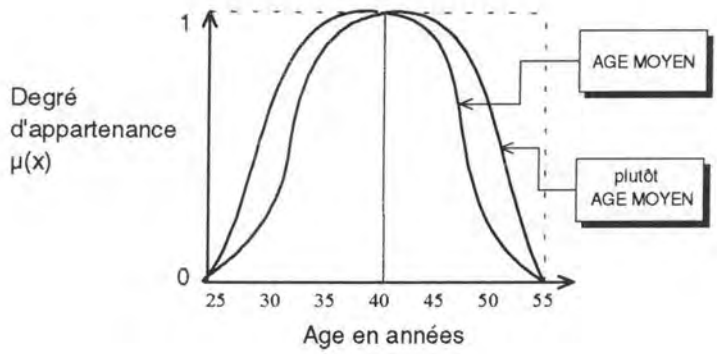

Figure 2.15 : Transformateur (plutôt) qui atténue la courbe de l'ensemble flou

De la même façon, il existe d'autres types de transformateurs. Le Tableau 2.1 nous donne un aperçu global de ces différents types de transformateurs [Cox, 94].

HEDGE	MEANING
about, around, near, roughly	Approximate a scalar
above, more than	Restrict a fuzzy region
almost, definitely, positively	Contrast intensification
below, less than	Restrict a fuzzy region
vicinity of	Approximate broadly
generally, usually	Contrast diffusion
neighbouring, close to	Approximate narrowly
not	Negation or complement
quite, rather, somewhat	Dilute a fuzzy region
very, extremely	Intensify a fuzzy region

Tableau 2.1: Les transformateurs linguistiques flous et leur signification

2.4.2. Signification et interprétation des transformateurs

Le mécanisme derrière les opérations de transformation est généralement de nature heuristique, c'est-à-dire que la façon dont la surface floue est transformée et la nature de la transformation ne sont pas basées sur des théories mathématiques, mais plutôt sur des critères de jugement.

L'originale définition de Lotfi Zadeh du transformateur *très*, par exemple, intensifie l'espace flou en mettant au carré la fonction d'appartenance de chaque point de l'ensemble ($\mu^2_A[x]$). De la même manière, le transformateur *plutôt* atténue l'espace flou en prenant la racine carrée de la fonction d'appartenance de chaque point de l'ensemble ($\mu^{1/2}_A[x]$). Ce choix est arbitraire et peut être modifié si nécessaire (exemple pour *très* : $\mu^3_A[x]$).

Les transformateurs jouent le même rôle dans la théorie des ensembles flous que les adjectifs ou les adverbes dans le langage naturel. L'application d'un transformateur change le comportement de la logique de la même manière que les adjectifs changent la signification dans les phrases humaines. Ainsi, on peut facilement représenter, par un ensemble flou et à partir de l'ensemble GRAND, la signification : *très* GRAND ou *pas* GRAND.

Comme les transformateurs ont une nature linguistique, on peut en appliquer plusieurs sur une simple région floue pour que celle-ci corresponde mieux aux caractéristiques sémantiques qu'elle veut représenter :

- Très très GRAND.
- Presque pas très GRAND.
- Généralement autour de GRAND.

2.5. Le raisonnement flou

2.5.1. Les variables linguistiques

Au centre de la technique de la modélisation floue se trouve l'idée de variables linguistiques. Dans son état le plus simple, une variable linguistique est le nom d'un ensemble flou. A la Figure 2.1, l'ensemble flou LONG est une simple variable linguistique et peut être utilisé dans un système de règles basé sur la longueur d'un projet particulier :

Si *durée.projet* est **LONG**

alors *risque* est **GRAND**

Cependant, les variables linguistiques peuvent aussi inclure les concepts de transformateurs (*hedges*) d'ensembles flous. Comme nous l'avons vu dans la partie précédente, ces transformateurs changent la courbe de l'ensemble :

Si *durée.projet* est **presque pas très LONG**

alors *risque* est **plutôt REDUIT**

Une variable linguistique encapsule les propriétés des concepts imprécis ou approximatifs. Elle est la représentation d'un ensemble flou (modifié ou non par les transformateurs).

2.5.2. Les propositions floues

Un modèle flou consiste en une série de propositions floues conditionnelles ou inconditionnelles. C'est l'ensemble de ces propositions qui va créer, au fur et à mesure, l'ensemble flou *solution*.

Une proposition conditionnelle est de la forme : *Si w est Z alors x est Y*, où Z et Y sont des variables linguistiques (c'est-à-dire des ensembles flous) et où w et x sont des valeurs prises dans les domaines de Z et Y respectivement. On peut interpréter cette proposition de la manière suivante : « *x est membre de Y au même degré d'appartenance que w est membre de Z* ».

La proposition conditionnelle peut être étendue de la manière suivante : *Si (w est Z)* ¤ *(y est V)* ¤ ... ¤ *(u est S) alors x est Y*, où ¤ est un opérateur de la forme ET ($\cap$) ou OU ($\cup$). Dans ce cas-ci, la position de x dans Y est déterminée par la composition de l'ensemble des valeurs situé avant le *alors*.

Une proposition inconditionnelle est de la forme : *x est Y*, où Y est une variable linguistique (c'est-à-dire un ensemble flou) et x une valeur dans le domaine de Y. On peut interpréter cette proposition de la manière suivante : « *x est le minimum sous-ensemble de Y* ». Lorsque l'ensemble flou de solution est vide, alors x est restreint à l'ensemble Y; dans le cas contraire, x devient le minimum entre l'ensemble flou solution déjà existant et l'ensemble Y. (L'ensemble flou *solution* est ainsi mis à jour en prenant l'intersection de l'ensemble *solution* et l'ensemble Y).

2.5.3. La technique de modélisation

Le mécanisme de base dans un modèle flou repose sur les propositions. Ce sont les relations entre les variables du modèle et une ou plusieurs régions floues. L'ensemble de ces propositions conditionnelles et inconditionnelles contribue à la création de l'ensemble flou *solution*. Le lien fonctionnel entre les degrés d'appartenance dans les régions floues est appelé *method of implication* (crée l'ensemble flou *solution*). Le lien fonctionnel entre les régions floues (représenté par l'ensemble flou de solution) et la valeur recherchée (solution du problème) est appelée *method of defuzzification*. Prises ensembles (*The fuzzy compositional rules of inference*), ces deux méthodes constituent l'épine dorsale du raisonnement flou.

Notre but n'étant pas de décrire en profondeur la théorie des ensembles flous mais plutôt d'en avoir une connaissance de base, nous n'allons pas expliquer en quoi consistent ces deux méthodes (le lecteur intéressé pourra trouver plus d'informations dans [Cox, 94]).

Prenons simplement, pour illustrer le principe du raisonnement flou, un exemple exposant une technique d'implication floue de base (*monotonic proportional reasoning*). C'est le cas où deux ensembles flous sont mis en rapport avec une simple proposition conditionnelle :

Si x est Y alors z est W

Considérons comme données de l'exemple :

- deux ensembles flous : GRAND et LOURD (définis à la Figure 2.16),
- la proposition : "*Si hauteur est GRAND alors poids est LOURD*"
- une hauteur (taille), soit 170 cm.

Tandis que la valeur recherchée est *poids*.

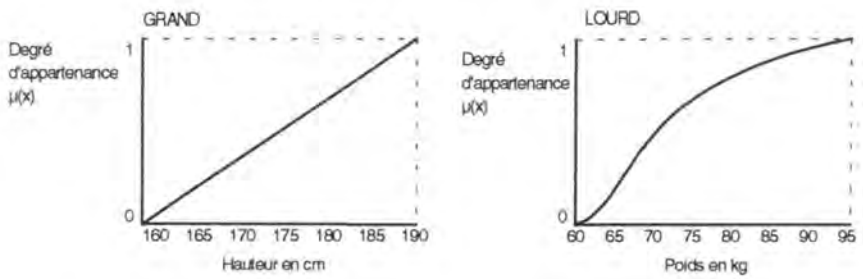

Figure 2.16 : Deux ensembles flous : GRAND et LOURD

La *monotonic proportional reasoning* se décompose en deux étapes :

- Pour un élément x dans le domaine de Y, trouver son degré d'appartenance dans la région floue Y (μ[x]), c'est-à-dire dans notre exemple, pour un élément hauteur, soit 170 cm, son degré d'appartenance à GRAND est de 0.37 (=μ[170cm]).
- Dans la région floue W au degré d'appartenance correspondant à μ[x] (=0.37), trouver le point du domaine correspondant (z), c'est-à-dire dans notre exemple 69 kg.

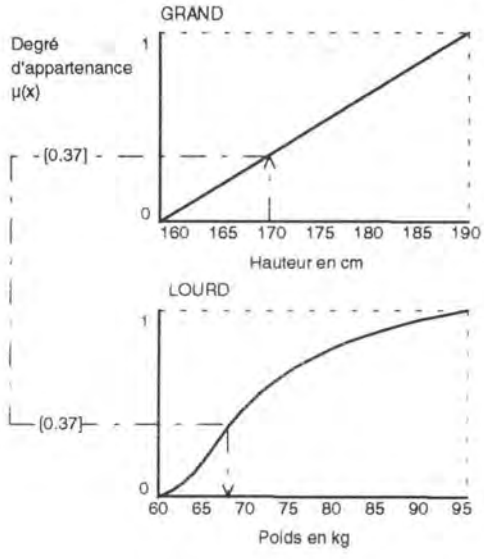

Figure 2.17 : Un exemple de méthode Monotonic Proportional Reasoning : Modèle d'estimation de poids

Comme le montre la Figure 2.17, la solution de notre problème donne : *poids* = 69 kg. Cet exemple correspond bien à l'idée que l'on se fait d'une proposition conditionnelle, c'est-à-dire que "*poids* est membre de LOURD au même degré d'appartenance que *hauteur* est membre de GRAND".

Chapitre 3

Les systèmes prédictifs de communication

Nous avons vu au premier chapitre qu'il existait plusieurs types de systèmes de prédiction. Il en existe au niveau des lettres, des mots et des phrases. Dans ce chapitre, nous allons passer en revue quelques logiciels d'aide à la communication basés sur ces systèmes de prédiction, le but étant d'évaluer ce qui existe actuellement sur le marché.

Le premier système que nous allons étudier est un système basé sur la prédiction des mots. Il accélère l'utilisation d'un traitement de texte pour des personnes peu expérimentées ou ayant des problèmes moteurs/mentaux. On verra également que l'on peut ajouter une série d'outils à ce système, lui permettant d'avoir une plus grande efficacité.

Nous étudierons ensuite deux systèmes de prédiction de petits textes ou de phrases. Ces systèmes permettent d'augmenter considérablement la vitesse de conversation. Ils demandent néanmoins une charge préliminaire plus importante pour l'utilisateur.

Nous terminerons ce chapitre par l'étude d'un système de communication basé encore sur la prédiction de phrases, mais aussi sur la théorie des ensembles flous. Nous verrons que ce prototype est le point de départ du projet FuzzyChat, que nous étudierons dans les prochains chapitres.

3.1. Word Prediction : PAL

3.1.1. Introduction

Ce projet se présente comme une prothèse de communication aidant des personnes qui ont des problèmes physiques et/ou mentaux à communiquer. Il utilise un système basé sur la prédiction de mots.

The *Predictive Adaptive Lexicon* (PAL) [Newell, 91] est un système de prédiction de mots développé au MicroCentre de l'université de Dundee (Ecosse). Au départ, PAL était destiné à aider les personnes ayant des handicaps physiques dans le cadre de l'utilisation d'un traitement de texte. Cela leur permettait d'accélérer l'utilisation de celui-ci en incorporant des prédictions précises de mots. Cependant, les bénéfices de l'utilisation d'un système prédictif ont prouvé que PAL pouvait être employé dans un contexte plus large, incluant des utilisateurs qui souffrent de maladie mentale.

PAL a été construit pour réduire le nombre de touches qu'une personne doit presser pour produire du texte lors de l'utilisation d'un traitement de texte.

3.1.2. Fonctionnement du système

PAL s'exécute sur ordinateurs IBM et compatibles, comme un programme « résident en mémoire ». Cela signifie qu'il peut s'exécuter en collaboration avec d'autres logiciels standards comme par exemple un traitement de texte.

PAL assiste l'utilisateur, *qui crée du texte*, en lui offrant un petit menu de prédiction de mots et en lui permettant de choisir parmi ces prédictions (Figure 3.1), par une simple pression d'une touche du clavier.

Le système PAL s'exécute, en général, en 'collaboration' avec un traitement de texte et fonctionne de la façon suivante (Figure 3.2) : l'utilisateur qui presse sur une touche, se voit proposer cinq mots par le système. Ces mots commencent tous par la touche pressée par l'utilisateur. Soit celui-ci trouve, parmi les propositions, le mot qu'il cherche, et appuie sur la touche de fonction correspondante; soit il ne trouve pas le mot qu'il cherche et appuie sur la deuxième lettre de son mot.

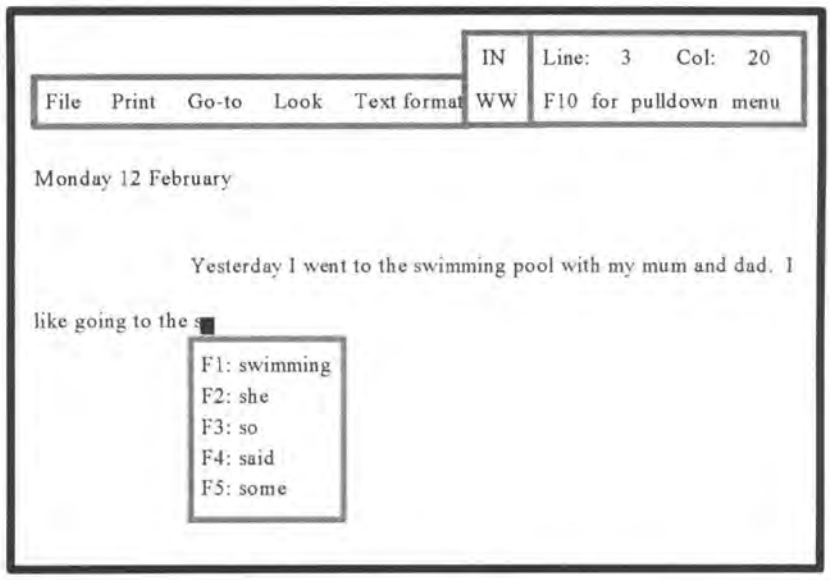

Figure 3.1 : Exemple d'utilisation de PAL dans un traitement de texte

Le système PAL produira alors une nouvelle liste de prédiction sur base des deux premières lettres du mot. Ce processus se répète jusqu'au moment où : soit le mot est proposé par le système, soit l'utilisateur a entré au clavier le mot entier.

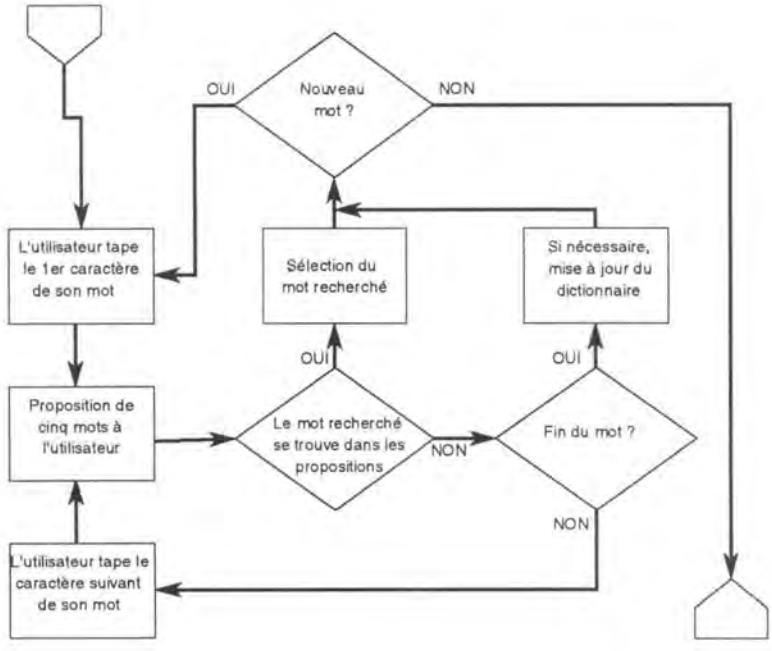

Figure 3.2 : Fonctionnement du système PAL

Les prédictions sont sélectionnées à partir d'un dictionnaire de mots, sur base de leur fréquence et de leur récente utilisation dans les précédentes exécutions du système.

PAL comprend deux dictionnaires standards, un de 1000 mots et l'autre de 5000 mots. Cependant, l'utilisateur pourra créer son propre dictionnaire ou compléter les dictionnaires existants.

Il est aussi possible d'insérer dans le dictionnaire des nouveaux mots durant l'exécution du programme. En effet, lorsque l'utilisateur doit entrer tout le mot au clavier (sans que le système ne puisse proposer ce mot) parce qu'il ne se trouve pas dans le dictionnaire, le système peut l'intégrer dans celui-ci automatiquement. Ainsi le mot pourra être proposé dans les futures prédictions. Cette possibilité peut être désactivée pour éviter le placement des mots mal orthographiés dans le dictionnaire. En effet, PAL est un système qui s'adresse notamment à des classes de jeunes écoliers en difficulté.

Comme nous l'avons vu, PAL peut être utilisé avec des traitements de texte différents. PALSTAR en est un particulièrement adapté à l'utilisation de ce système. Il donne notamment la possibilité d'incorporer l'usage d'un synthétiseur de voix au système. Ainsi l'utilisateur peut écouter ce qu'il a écrit, permettant à certaines personnes, qui ont des difficultés de lecture, d'entendre leurs erreurs.

3.1.3. Evaluation du système PAL

Quatre années intensives de tests ont donné la preuve que le système PAL peut aider énormément dans le domaine de l'apprentissage et le développement du langage. Utilisé avec un traitement de texte, il propose des prédictions à chaque lettre tapée. Il réduit le nombre de pressions de touches d'environ 50 % [Ricketts, 91]. De plus, lors de son utilisation, PAL produit (ou complète) le dictionnaire courant en stockant dans celui-ci les noms et mots propres à l'utilisateur. Même les étudiants ayant de sérieux problèmes de lecture peuvent arriver à des résultats gratifiants.

PAL a fait l'objet d'amples tests dans des écoles, dans des homes et dans des centres d'éducation pour adultes en Europe et aux U.S.A., avec des résultats remarquables.

PAL peut être particulièrement utile pour :

- Les enfants de tout âge.
- Les étudiants ayant comme seconde langue l'anglais.
- Les étudiants en langue étrangère utilisant les caractères romains (français, italien, allemand, etc).

- Les dactylographes lents ou inexpérimentés.
- Les dyslexiques et les personnes ayant des problèmes de lecture.

PAL peut offrir une aide, entre autres, en ce qui concerne :

- La rapidité de génération de texte.
- La précision de l'orthographe.
- L'apprentissage de langues étrangères.

3.1.4. Syntax PAL

Pour certains utilisateurs, l'augmentation de production de texte obtenue grâce au système PAL fait apparaître d'autres problèmes linguistiques : les structures des phrases sont fausses et les mots sont syntaxiquement incorrects. Pour remédier à cela, un groupe de chercheur a créé une extension au système PAL, Syntax PAL, qui utilise le contexte de la phrase et des règles de grammaire pour effectuer ses prédictions [Morris, 92].

Syntax PAL tient compte encore de la fréquence des mots dans son algorithme de prédiction ainsi que de ses utilisations récentes, mais il exécute, en plus, une complète analyse syntaxique de la partie de phrase déjà tapée. Il proposera donc plus vite les mots syntaxiquement corrects que les mots incorrects.

3.1.5. PAL & SPELLER

SPELLER est un correcteur orthographique, construit dans le but de corriger les fautes d'orthographes (provenant notamment de problèmes dyslexiques). Lorsqu'une faute est rencontrée, une liste de cinq suggestions est alors proposée pour que l'utilisateur choisisse le mot correct. La liste des suggestions est basée sur une analyse lexicale et phonétique du mot incorrect. Cela diffère de la plupart des systèmes commerciaux existant sur le marché, qui se basent seulement sur une analyse lexicale.

L'analyse phonétique permet au système de suggérer des mots ayant des sons identiques (exemple : "physique" est suggéré comme correction de "fisic"). Lors des tests réalisés chez des enfants atteints de ce type de problèmes, SPELLER a démontré une efficacité significative par rapport aux correcteurs orthographiques classiques.

PAL & SPELLER ont été combinés dans un programme qui permet à l'utilisateur de bénéficier du système de prédiction de mots, complété par le système de correction orthographique.

3.2. Sentence Prediction : PROSE & TALKSBAC

3.2.1. Introduction

Plutôt que de faire des prédictions de mots, certains systèmes font des prédictions de phrases, voire même de petits textes. PROSE et TALKSBAC en sont deux exemples. Ils se basent sur le fait qu'en général, nous avons des sujets de prédilection dans nos conversations. Ainsi de retour de vacances, on a souvent envie de raconter comment elles se sont passées. De même, il n'est pas rare que l'on explique plusieurs fois notre situation familiale ou professionnelle durant des conversations différentes.

De ce fait, ces systèmes stockent au préalable les informations propres à l'utilisateur et les reproposent lors de leurs utilisations. Cela demande, bien évidemment, un effort non négligeable à l'utilisateur (ou à son entourage) puisqu'il doit fournir au système l'ensemble de ses données avant de pouvoir l'utiliser.

Néanmoins, ces systèmes permettent d'alléger la tâche de l'utilisateur lors de la conversation, en utilisant des liens sémantiques dans la prédiction de phrases. Ainsi, la charge de recherche d'information se trouve sur le système plutôt que sur l'utilisateur.

3.2.2. Predictive Retrieval of Story Extracts (PROSE)

Predictive Retrieval of Story Extracts (PROSE) est donc un système de communication permettant à l'utilisateur de créer une sortie parlée à partir d'un synthétiseur de voix [Alm, 93a]. Il se base sur un système de prédiction de phrases ou de morceaux de texte. Cette prédiction se fait à partir d'une base de données personnelles et d'un minimum d'informations entrées par l'utilisateur lors de l'exécution du logiciel.

Le fonctionnement du système PROSE se fait de la façon suivante : au départ, l'utilisateur choisit un sujet particulier (vacances, sport, famille, voyage, etc). Il doit ensuite choisir les trois perspectives qu'aura sa conversation :

- **Personne** : 'me' ou 'you' .

- **Temps** : 'past', 'present' ou 'future'.
- **Orientation** : 'where', 'what', 'how', 'when', 'who' ou 'why'.

Pour chaque combinaison de perspectives et pour chaque sujet, le système proposera 15 phrases ou petits textes à l'utilisateur. Celui-ci n'aura plus qu'à sélectionner les phrases pour que le synthétiseur de voix les *parle*. Il pourra également changer de perspectives ou de sujets en cours de conversation.

Il faut néanmoins que l'utilisateur entre ses données personnelles avant de pouvoir utiliser le système. Il devra les rentrer en indiquant pour quelles perspectives et quels sujets, elles sont destinées.

3.2.3. TALKSBAC

Talksbac assiste les utilisateurs, pour créer une communication interactive, en offrant des choix d'expressions conversationnelles (phrases ou titres d'histoire) à l'écran ([Broumley, 90] et [Waller, 91]). Une fois sélectionnés, ces choix sont *parlés* par un synthétiseur de voix. C'est le système qui propose ces expressions, en apprenant au fur et à mesure ce que le client voudrait dire.

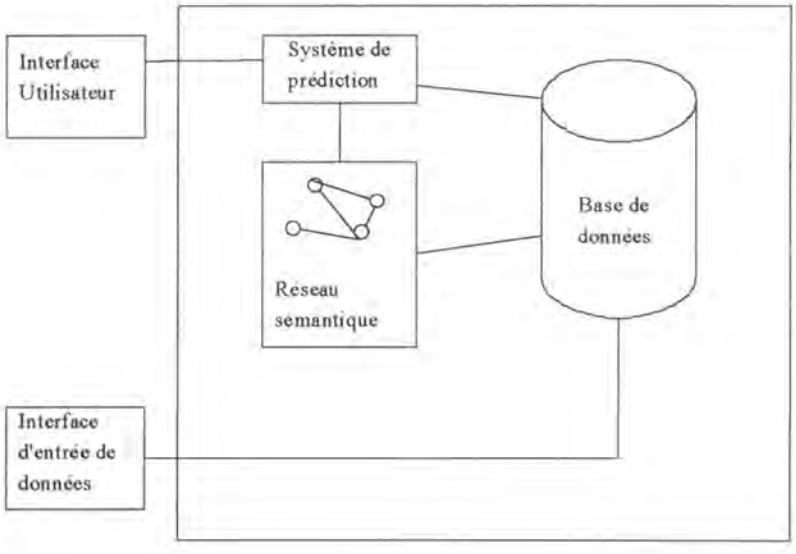

Figure 3.3 : Différents modules du système TALKSBAC

Talksbac comporte une base de données de phrases et d'histoires, un système de prédiction, un réseau sémantique, une interface utilisateur et une interface d'entrée de données (Figure 3.3).

L'interface d'entrée de données permet à l'utilisateur d'introduire dans le système ses nouvelles données (phrases ou histoires) et de supprimer ou modifier ses données existantes.

L'interface utilisateur permet de manipuler le système au cours de son exécution. Le premier choix de l'utilisateur sera de sélectionner le type de personnes avec lequel il parle. Il doit ensuite choisir le sujet sur lequel il veut s'entretenir. Le système de prédiction lui proposera alors en fonction de cela des phrases ou des titres d'histoire.

Le système de prédiction utilise des techniques tirées de l'intelligence artificielle pour faire ses propositions. Chaque phrase ou histoire de la base de données est associée avec un nombre fini d'informations (Figure 3.4). C'est sur base de ces informations que le système fera ses prédictions à l'utilisateur.

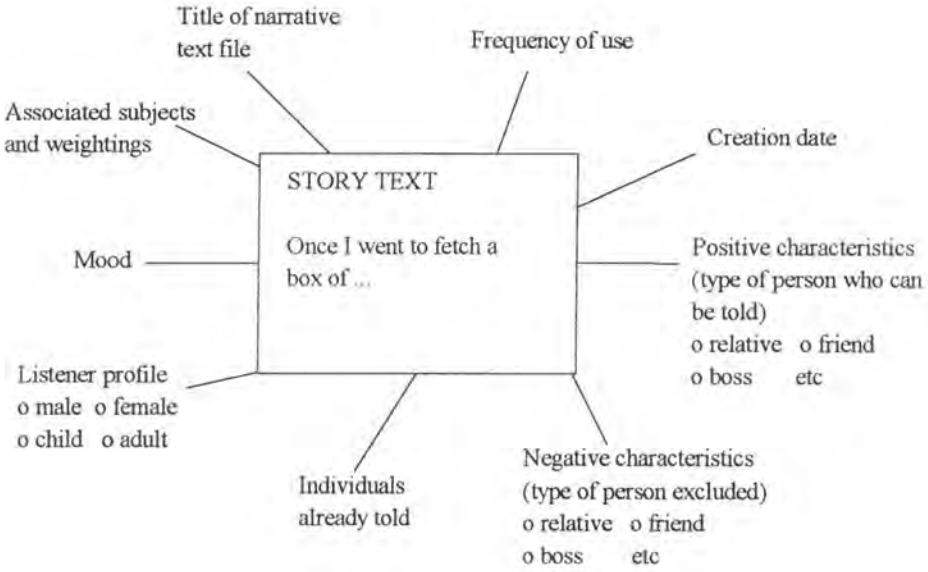

Figure 3.4 : Informations associées à chaque phrase de la base de données TALKSBAC

3.3. TOPIC : utilisation d'un algorithme flou de recherche

3.3.1. Introduction

Un autre système de prédiction de phrases existe mais n'est encore qu'à l'état de prototype. C'est le système TOPIC [Nicol, 93]. Il est basé sur la théorie des ensembles flous et permet, comme les autres, de faire des prédictions en se basant sur une base de don-

nées de phrases préexistantes. En prenant comme référence la phrase courante, il va rechercher dans la base de données, les trois phrases qui sont les plus proches au point de vue du contenu (Figure 3.5). Ainsi, le système va fournir à l'utilisateur la possibilité d'élaborer une conversation aussi naturelle que possible.

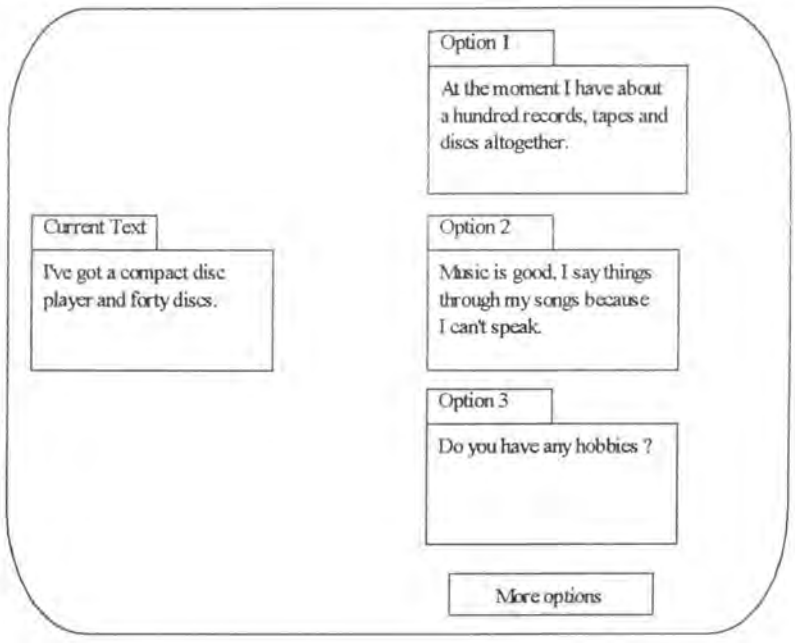

Figure 3.5 : Exemple d'exécution de TOPIC

Il utilise aussi, comme les autres systèmes, un synthétiseur de voix permettant de *parler* les phrases sélectionnées.

3.3.2. Le système de prédiction TOPIC

Détaillons le système de prédiction de TOPIC. Comme les précédents systèmes, l'utilisateur doit au préalable remplir la base de données du logiciel. Celle-ci est constituée de deux éléments : de phrases et de descripteurs. A chaque phrase placée dans la base de données, l'utilisateur doit l'associer avec un nombre limité de descripteurs. Ces descripteurs sont de deux types : descripteurs de sujets et descripteurs de buts. Les descripteurs de sujets associent la phrase avec des sujets bien précis (exemple : sport, vacances, loisir, travail, musique, etc). Les descripteurs de buts associent la phrase avec les *temps* de la conversation (exemple : introduction, élaboration, conclusion, etc).

Expliquons cela par un exemple. Considérons les descripteurs de sujets : *sport, vacances* et *travail*; considérons les descripteurs de buts : *introduction* et *élaboration*; et

considérons la phrase *« J'ai réussi mon brevet de 800 mètres en natation »*. L'utilisateur doit donc associer cette phrase à chaque descripteur existant, en assignant une valeur entre 1 et 20. L'utilisateur assignera par exemple pour le descripteur :

- **'Sport'** une valeur de 18, car la phrase est fort associée au sujet *sport*.
- **'Vacances'** une valeur de 10, car la phrase peut être dans une certaine mesure associée à l'idée de vacances.
- **'Travail'** une valeur de 2, car la phrase ne correspond pas du tout au sujet *travail*.
- **'Introduction'** une valeur de 7, car la phrase va rarement introduire une conversation.
- **'Elaboration'** une valeur de 17, car la phrase est fortement considérée comme une phrase qui élabore un sujet.

Ainsi, chaque phrase de la base de données sera associée aux descripteurs présents dans le système. Cette association est subjective et dépend bien sûr de chaque utilisateur. Elle peut être aussi considérée comme une association à un ensemble flou. En effet, on peut considérer qu'elle donne un degré d'appartenance aux phrases de la base de données pour chaque descripteur (considéré comme un ensemble flou).

Le système de prédiction TOPIC fonctionne de la façon suivante : pour chaque prédiction, il compare les descripteurs de la phrase courante avec les descripteurs de toutes les autres phrases dans la base de données. Cette comparaison se fait par une simple différence arithmétique (entre descripteurs). Les phrases proposées seront celles qui auront le moins de différence avec la phrase courante. Ainsi, le système *prédira* les phrases qui sont les plus proches en contenu de cette phrase courante.

3.3.3. Le projet FuzzyChat

Le projet FuzzyChat, décrit d'une manière approfondie dans la seconde partie, se base sur le système TOPIC, pour créer un système flou de prédiction et de recherche d'information. En effet, il reprend l'idée d'associer des descripteurs aux phrases de la base de données. Cependant, FuzzyChat est un système qui a été entièrement refait, en incluant :

- Un algorithme flou plus complexe qui tient compte notamment de la fréquence d'utilisation des phrases.

- Une interface graphique adaptée aux systèmes de communication.
- Des options et des fonctions supplémentaires permettant de tester le système dans un cadre aussi naturel que possible.

Le but de ce projet est de continuer à explorer la possibilité d'appliquer la théorie des ensembles flous dans la construction d'un système d'aide à la communication pour les personnes souffrant de problèmes de langage.

Partie II

Création d'un système flou d'aide à la communication : FuzzyChat

Chapitre 4

Développement d'un système flou de recherche d'information

Dans ce chapitre, nous allons expliquer le développement d'un nouveau système de recherche d'information, permettant d'effectuer des prédictions dans une prothèse de communication. Ce système se base sur la théorie des ensembles flous.

Après avoir détaillé l'ensemble des fonctions et des possibilités du logiciel Fuzzy-Chat, nous allons passer en revue les différents périphériques d'entrée et de sortie utilisés dans ce système. De la même façon, nous détaillerons les différents fichiers d'entrée et de sortie dont le système a besoin pour fonctionner.

Pour clôturer ce chapitre, nous expliquerons en détail comment fonctionne l'algorithme flou de prédiction du logiciel.

Nous ne donnerons ici aucune explication, ni justification de l'interface du système, ceci étant le but du chapitre suivant.

4.1. Développement d'un nouveau système de communication

4.1.1. Données de base

Une des méthodes, en vue d'assister certaines personnes à communiquer plus rapidement, consiste en l'utilisation de systèmes qui offrent la possibilité de sélectionner des phrases complètes ou des morceaux de texte, plutôt que des mots ou des lettres. Néanmoins, la difficulté de cette approche réside dans les moyens d'accéder à ces phrases ou morceaux de texte, sans pour cela ralentir la vitesse de la conversation.

D'autre part, il est aussi important de remarquer que, dans une conversation, les participants ne passent pas sans arrêt du *coq à l'âne*. Ils parlent généralement d'un même sujet durant un petit temps avant d'en changer.

Un système de prédiction, se basant sur le contenu de phrases stockées et s'occupant de la recherche des phrases dans la base de données, permettrait d'obtenir une conversation plus naturelle, plus réaliste et plus rapide.

4.1.2. Spécification des exigences

Voici, de manière concise, l'ensemble des exigences requises au départ du projet. Elles ont été déterminées après plusieurs discussions avec des chercheurs et utilisateurs de prothèses de communication :

- Développement d'un algorithme flou de recherche d'information, créant un nouveau type de système de communication. Le système doit retrouver les quatre phrases (dans la base de données) les plus proches sémantiquement de la phrase principale.
- Développement d'une interface adaptée pour un système de communication.
- Création de l'option « More » : provoque une nouvelle recherche dans la base de données.
- Création de l'option « Freeze » / « Unfreeze » : active ou désactive le système de prédiction.
- Création des options de *backtracking* (les boutons « < » et « > »).
- Création d'une sortie parlée à partir d'un synthétiseur de voix.
- Exécution du système sur PC.

Si le temps le permettait (et ce fut le cas), les deux idées suivantes devaient être implémentées pour donner au projet plus d'ampleur :

- Possibilité de créer manuellement du texte à partir du clavier.
- Possibilité de faire de petits commentaires rapides et répétitifs.

4.2. Le projet : FuzzyChat

Rappelons-le : le but de ce travail est d'explorer la possibilité d'appliquer la théorie des ensembles flous dans la création d'un système de communication, pour des personnes atteintes de problèmes de langage. Ce projet s'appuie sur le travail décrit dans la partie « *3.3. TOPIC : utilisation d'un algorithme flou de recherche* » et prolonge ses recherches en appliquant des algorithmes flous plus complexes et en créant une nouvelle interface.

C'est la personne, ayant des difficultés de parole, qui utilise ce logiciel pour dialoguer avec une autre personne. Ce système essaye d'être le plus naturel possible en créant une sortie parlée sonore à l'aide d'un synthétiseur de voix.

Ce projet est un système de prédiction de phrases de conversation. Il relie ces phrases entre elles, en appliquant une méthode de recherche d'information floue. L'utilisation de techniques de prédiction dans un système de communication a pour but d'accélérer la vitesse de conversation.

Ce type de méthode permet de retrouver les phrases (dans la base de données) qui sont les plus proches sémantiquement de la phrase principale. De cette manière, le système essaye de prédire les prochaines phrases que l'utilisateur voudrait dire. La responsabilité de la recherche d'information n'est pas à charge de l'utilisateur mais à charge du système lui-même. Cela permet à la personne handicapée de participer plus facilement et plus naturellement à la conversation.

En plus du système de prédiction (auquel s'applique la théorie des ensembles flous), le logiciel inclut la possibilité de créer manuellement du texte au clavier. En effet dans un système tel que celui-ci, on ne peut pas stocker dans la base de données toutes les phrases possibles. Comment, par exemple, deviner les réponses aux questions de notre partenaire, sans connaître ses questions ? Il est donc nécessaire d'inclure la possibilité (au risque de ralentir la conversation) de taper au clavier les mots ou phrases que l'utilisateur voudrait dire.

De même, le système permet à l'utilisateur de *dire*, via le synthétiseur de voix, des petits commentaires rapides d'approbation ou de remerciement. En effet, de nombreuses petites phrases reviennent toujours dans une conversation et rendent celle-ci plus vivante. Voici les petits commentaires disponibles dans ce système :

- *Hello, how are you ?*
- *I'm fine, thank you !*
- *Well, that is about it.*
- *Good bye.*
- *Yes.*
- *No.*
- *I agree.*
- *I disagree.*
- *That is good.*
- *That is bad.*
- *I don't know.*
- *Thank you.*
- *Please, wait while I find the words.*
- *Please, tell me more about it.*
- *Sorry !*
- *Uh-huh.*

Enfin, le système FuzzyChat inclut cinq options permettant une plus grande souplesse dans l'utilisation du logiciel. Ces options sont :

- **More** : provoque une nouvelle recherche dans la base de données pour trouver les quatre phrases suivantes qui sont les plus proches sémantiquement de la phrase principale. Cette option permet à l'utilisateur, n'ayant pas trouvé ce qu'il cherchait dans les propositions affichées à l'écran, de demander une nouvelle recherche dans la base de données. Le système propose alors quatre autres phrases à l'utilisateur.
- **Freeze/Unfreeze** : désactive ou réactive le système de prédiction. L'utilisateur peut avoir envie de *dire* (via le synthétiseur de voix) plusieurs phrases proposées sans que le système ne mette à jour l'espace de prédiction.

- **< et >** : fournit un outil de *backtracking* (permet à l'utilisateur d'aller et venir dans les écrans précédents).
- **Help** : fournit deux écrans d'aide guidant les nouveaux utilisateurs.
- **Exit** : quitte le système.

En prenant l'ensemble des fonctions et des options de ce système, FuzzyChat essaye de s'inscrire dans un système global et complet de communication. Il veut donner les moyens de créer une conversation aussi réaliste que possible.

4.3. Les périphériques d'entrée et de sortie du système

La Figure 4.1 nous montre quels sont les types de périphériques utilisés dans le logiciel. Il y a deux périphériques d'entrée et deux périphériques de sortie.

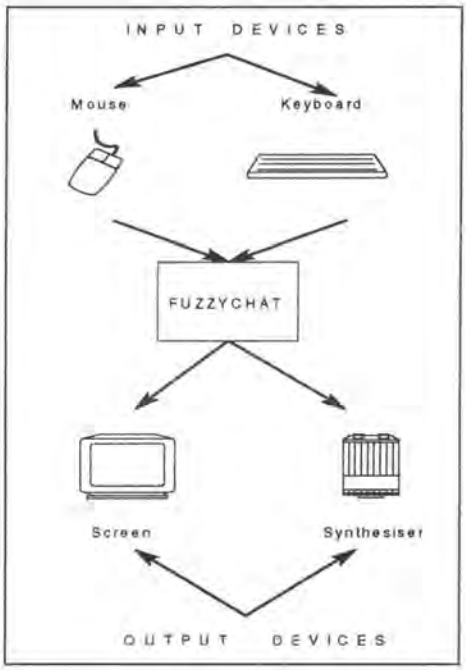

Figure 4.1 : Périphériques d'entrée et de sortie du logiciel

Le principal périphérique d'entrée est la souris. C'est grâce à elle que l'utilisateur interagit avec le système tout le long de la conversation. Le clavier ne sert que lorsque l'utilisateur veut créer manuellement du texte.

Il est important ici de remarquer que l'usage de la souris n'est pas trop restrictif. En effet, la personne handicapée ne sachant pas manier la souris pourrait faire appel à des

variantes telles que le *joystick*, la souris au casque, etc... De plus, il existe sur le marché plusieurs logiciels permettant à la personne moins valide d'exercer un contrôle sur l'interface d'un système. Ce contrôle peut s'effectuer de différentes manières : par la voix, par un écran tactile, par un *push button*, par le clavier, etc... [Gardeazabal, 95]. Par exemple, la technique de balayage[1] de l'écran peut être utilisée avec un *push button*. Ainsi, l'utilisateur pressera le bouton au moment où le logiciel sélectionnera l'objet interactif voulu.

L'écran et le synthétiseur de voix sont les deux périphériques de sortie du logiciel. L'écran sert, entre autres, à donner un bon *feedback* à l'utilisateur. Le synthétiseur de voix permet d'entendre les phrases sélectionnées pendant l'exécution du système (c'est-à-dire que le synthétiseur *parle* les phrases sélectionnées).

4.4. Les fichiers d'entrée et de sortie

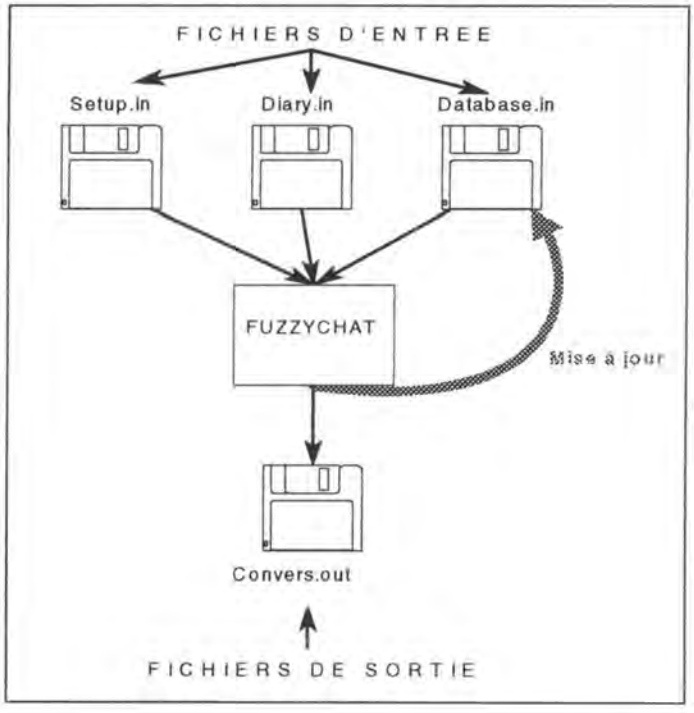

Figure 4.2 : Les fichiers d'entrée et de sortie du logiciel

La Figure 4.2 nous montre les différents fichiers utilisés dans le système. Il y a trois fichiers d'entrée ('Setup.in', 'Diary.in' et 'Database.in') et un fichier de sortie

[1] Logiciel KENIX fonctionnant sur Mac Intoch et fabriqué par Don Johnstone.

('Convers.out'). De plus, le fichier 'Database.in' est mis à jour à la fin de la session FuzzyChat. Tous ces fichiers sont de simples fichiers textes. Chaque utilisateur devra créer ses propres fichiers avec ses données propres, avant de pouvoir utiliser ce logiciel. Ce sont des fichiers personnels.

Seul le fichier 'Database.in' est absolument nécessaire pour faire tourner le système. En l'absence des fichiers 'Setup.in' et 'Diary.in', des valeurs par défaut seront prises en compte.

4.4.1. Le fichier d'entrée : 'Setup.in'

Il y a dans ce fichier deux options que l'utilisateur peut changer, concernant le synthétiseur de voix. La Figure 4.3 nous montre un exemple de ce fichier.

```
COM         2

SYNTH       ON
```

Figure 4.3 : Exemple de fichier 'Setup.in'

La première ligne indique sur quelle sortie est branchée le synthétiseur de voix. Il peut y avoir deux connexions différentes : soit port1, soit port2. Dans le cas du port1, la valeur après COM devra être '1'; dans le cas de port2, la valeur après COM devra être '2'.

La seconde ligne du fichier permet de brancher ou de débrancher le synthétiseur de voix. Si la valeur après le SYNTH est 'ON', le synthétiseur est branché; si elle est 'OFF', il est débranché.

4.4.2. Le fichier d'entrée : 'Database.in'

C'est dans ce fichier que se trouvent stockées toutes les phrases que l'utilisateur pourrait ou voudrait dire (c'est la base de données du système). A chaque phrase est associé un nombre limité de descripteurs. Il y a quatre types de descripteurs : les descripteurs de sujets, de buts, de personnes et de fréquence. La Figure 4.4 nous montre un exemple d'une phrase dans un fichier 'Database.in' avec ses descripteurs.

Les trois premiers types de descripteurs (sujets, buts et personnes) permettent au système de faire des liens sémantiques entre les différentes phrases de la base de données :

- **Les descripteurs de sujets** : ils sont utilisés pour indiquer le degré de pertinence de la phrase en question par rapport à différents sujets. Voici quelques exemples de sujets utilisés dans le système actuel : *travel, hobbies, sport, music, driving, party, work, family, ...*

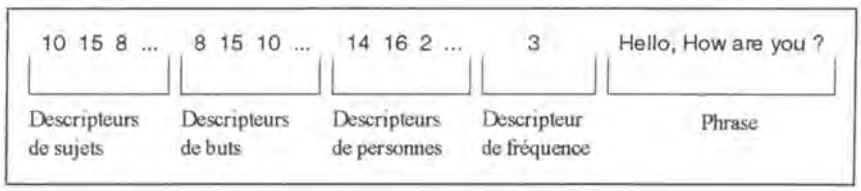

Figure 4.4 : Une ligne du fichier 'Database.in'

- **Les descripteurs de buts** : ils sont utilisés pour indiquer le degré de pertinence de la phrase en question par rapport aux différents moments de la conversation : *introduction, elaboration, conclusion, joke, question, ...*
- **Les descripteurs de personnes** : ils sont utilisés pour indiquer le degré de formalité de la phrase en question. Ce degré de formalité dépend de la personne avec laquelle on parle (on ne parlera pas, par exemple, de la même façon à son patron qu'avec son meilleur ami). Voici quelques exemples de descripteurs de personnes : *friends, family, boss, colleagues, child, foreigner, stranger, ...*

Chacun de ces descripteurs a une valeur comprise entre 1 et 20, selon le degré de pertinence de la phrase avec son propre descripteur. La première tâche du futur utilisateur du système est de créer sa base de données en choisissant ses propres descripteurs sujets, buts et personnes et en les associant avec ses propres phrases. Le nombre maximum de descripteurs sujets est limité à 10, tandis que le nombre maximum de descripteurs buts et personnes est limité à 8.

Le quatrième descripteur (celui de fréquence) est utilisé pour indiquer le degré d'apparition de la phrase en question dans les précédentes conversations (si la phrase a été souvent sélectionnée ou non). Il permet d'attacher plus d'importance aux phrases souvent utilisées lors des prédictions. Cette valeur varie entre 1 et 5. Plus la phrase est souvent sélectionnée lors d'une conversation antérieure, plus son descripteur de fréquence sera proche de 5 et inversement dans le cas contraire.

```
8 4 7

work family party driving sport music hobbies travel joke opening elaboration
conclusion friends relative boss colleagues child stranger foreigner fre-
quency

10 10 10 10 10 10 10 10 2 18 10 5 15 15 10 12 15 12 12 1 Hello, how are you ?

10 10 10 10 10 10 10 10 2 18 10 5 15 15 8 12 14 10 10 2 Are you having a good
day ?

5 5 15 5 15 15 19 13 2 18 10 5 15 15 5 12 10 10 10 1 What are your hobbies ?

2 5 10 10 15 5 19 13 2 15 15 8 15 15 8 10 9 10 10 1 One of my hobbies is be-
ing in the boyscouts. I'm a boyscout chief.

2 8 10 10 15 5 19 13 2 15 15 8 15 15 8 10 5 8 8 1 I have been a chief for 5
years and I like it !

5 5 10 10 18 5 19 10 2 15 15 8 15 15 8 10 9 10 10 2 One of my hobbies is ice
skating.

5 8 5 10 18 5 19 10 2 15 15 8 15 15 8 10 9 10 10 1 One of my hobbies is swim-
ming.

5 8 5 10 18 5 19 10 2 18 10 5 15 15 8 12 12 10 10 2 Do you like to swim ?

5 13 10 13 12 12 15 19 2 15 15 8 15 16 10 12 8 14 14 2 I like to travel
around the world. My favourite journey was to Zaire.

5 8 12 5 5 5 2 2 2 2 8 18 16 15 5 10 10 10 10 2 I'm very tired today, I'm go-
ing to sleep a bit...

5 5 10 10 18 5 17 10 2 18 15 8 15 15 8 10 5 10 10 2 Do you like ice skating ?
Do you skate ?

2 5 10 10 15 5 15 16 2 12 15 8 15 15 8 10 9 10 10 1 What I like in the boy-
scouts is the camp at the end of the year.

18 5 10 12 5 5 10 8 2 15 15 8 15 13 14 14 5 11 12 5 I'm studying computer
science.

...
```

Figure 4.5 : Exemple d'un fichier 'Database.in'

Au lancement du système, celui-ci décharge ce fichier dans la base de données du système. A la fin d'une conversation, le système met à jour le descripteur de fréquence de chaque phrase du fichier. Il le fait de la façon suivante : pour chaque phrase dans le fichier :

- Si elle a été proposée par le système de prédiction, mais pas sélectionnée par l'utilisateur, son descripteur de fréquence sera diminué de 1 (sauf si sa valeur est égale à 1).
- Si elle a été proposée par le système de prédiction et sélectionnée par l'utilisateur, son descripteur de fréquence sera augmenté de 1 (sauf si sa valeur est égale à 5).
- Si elle n'a pas été proposée par le système de prédiction, il n'y aura pas de changement dans son descripteur de fréquence.

La Figure 4.5 nous montre un exemple de fichier 'Database.in'. La première ligne indique au système le nombre de descripteurs de sujets, buts et personnes. La deuxième

ligne rappelle l'ordre des différents descripteurs à l'utilisateur. A partir de la troisième ligne se trouvent toutes les phrases qui constitueront la base de données du système.

Ce fichier est le plus grand des trois fichiers d'entrée. Il implique pour l'utilisateur une grande charge préliminaire d'encodage avant qu'il ne puisse utiliser le système. Cette phase d'encodage peut être très lourde pour certains utilisateurs et nécessite de toute façon beaucoup de patience et de temps. Néanmoins, une fois que le fichier est rempli, l'utilisateur ne devra le modifier que de temps en temps (pour le mettre à jour), s'il le désire.

Pour faciliter ce travail d'encodage, j'ai utilisé une feuille de calcul de Microsoft Excel pour encoder mes données. Cela m'a permis de ne pas me tromper dans la pondération des différents descripteurs (grâce aux colonnes d'Excel). Cependant, il serait peut-être nécessaire de créer un logiciel mieux adapté à ce travail d'encodage, le rendant ainsi moins contraignant et astreignant.

4.4.3. Le fichier d'entrée : 'Diary.in'

Ce fichier permet de mettre plus de poids sur certains descripteurs lors de l'exécution de l'algorithme de prédiction. Cela part du principe qu'à certains moments de la journée, on a plus tendance à parler de certains sujets que d'autres. Ainsi, si le lundi matin, je fais du sport, j'aurai plus tendance à parler de sport à ce moment-là.

Ce fichier est divisé en deux parties :

- Une partie agenda hebdomadaire divisée en jours et heures de la semaine.
- Une partie agenda annuel où l'on indique certaines dates importantes.

Comme le montre la Figure 4.6, on peut associer les descripteurs sujets, buts ou personnes (ceux présents dans le fichier 'Database.in') à certaines heures de la semaine ou certaines dates de l'année. Lors de l'exécution de l'algorithme de recherche, ses descripteurs auront plus d'importance.

Prenons l'exemple de la Figure 4.6, le lundi entre 9 et 10 heures, les phrases associées aux sports et destinées à être parlées aux collègues, auront plus de chance d'être sélectionnées que les autres. De même, le 25 décembre, les phrases relatives aux fêtes en famille seront plus vite proposées à l'utilisateur.

```
Monday
0500
0600
0700
0800
0900 Sport  Colleagues
1000 Sport  Friends Child
1100 Work
1200
1300
...
1900
2000
2100 Music Friends
2200
2300
2400

Tuesday
...

Wednesday
...

Thursday
...

Friday
...

Saturday
...

Sunday
...

Date
1/1/95 party friends
31/5/95 work boss
21/7/95 friends hobbies music
25/12/95 party family
```

Figure 4.6 : Exemple de fichier 'Diary.in'

4.4.4. Le fichier sortant : 'Conversa.out'

Ce fichier est créé au fur et à mesure de l'utilisation du système. Il donne un historique de toutes les actions qui ont été exécutées durant une session FuzzyChat. Ce fichier est créé principalement pour estimer les performances du logiciel. Il permet, entre autres, d'évaluer la rapidité des actions de l'utilisateur (Figure 4.7).

Si ce fichier existe déjà dans le répertoire courant, les données sont rajoutées à la suite du fichier présent. Dans le cas contraire, le fichier est créé.

```
New conversation : Sun Nov 27 13:56:28 1994
    13:51:27    BOX 0 : Hello  how are you ?
    13:52:02    UNIQUE TEXT : Hello !!
    13:52:52    BOX 4 : What do you like watch on the television ?
    13:53:17    MORE BUTTON
    13:53:35    FREEZE BUTTON
    13:53:37    BOX 2 : What kind of music do you like ?
    13:53:42    UNFREEZE BUTTON
    13:53:43    BOX 3 : Do you like beer ?
    13:55:02    CHAT BOX
    13:55:10    < BUTTON
    13:55:12    < BUTTON
    13:55:30    > BUTTON
    13:56:05    BOX 1 : In Belgium,  there are lots of kinds of beer.
Almost every town has its own beer .
    13:57:32    EXIT BUTTON
```

Figure 4.7 : Exemple de fichier 'Conversa.out'

4.5. L'algorithme flou de prédiction

4.5.1. Introduction

C'est cet algorithme qui constitue le système de prédiction du logiciel. Il utilise la théorie des ensembles flous dans sa recherche d'information. Il doit, à partir de la phrase principale, trouver les 4 phrases qui sont plus proches sémantiquement de celle-ci. Il le fait en comparant les descripteurs de la phrase principale aux descripteurs de tous les autres éléments dans la base de données.

Cumulativement, ces descripteurs peuvent être vus comme un moyen de localiser une phrase à un point particulier dans un espace multidimensionnel. La comparaison de deux phrases se fait en comparant les positions de celles-ci dans cet espace multidimensionnel et en additionnant les différences.

L'algorithme de prédiction tient également compte de la fréquence d'utilisation de chaque phrase dans les précédentes conversations. De même dans une conversation, il ne reproposera pas directement les phrases qui viennent d'être proposées.

4.5.2. Comparaison de deux phrases

Comment faire la comparaison entre deux phrases de la base de données ? En établissant les différences des descripteurs des deux phrases. Pour chacun des descripteurs (excepté pour le descripteur de fréquence), on prend sa valeur dans la première phrase et dans la deuxième phrase. On obtient donc deux valeurs comprises entre 1 et 20, représentant la

pertinence de chacune des phrases par rapport au même descripteur. Pour l'exemple, nous allons comparer le descripteur de sujet *sport*.

Voici les différentes étapes que doit alors effectuer le système pour déterminer la différence (au sens où nous l'entendons ici) entre ces deux valeurs :

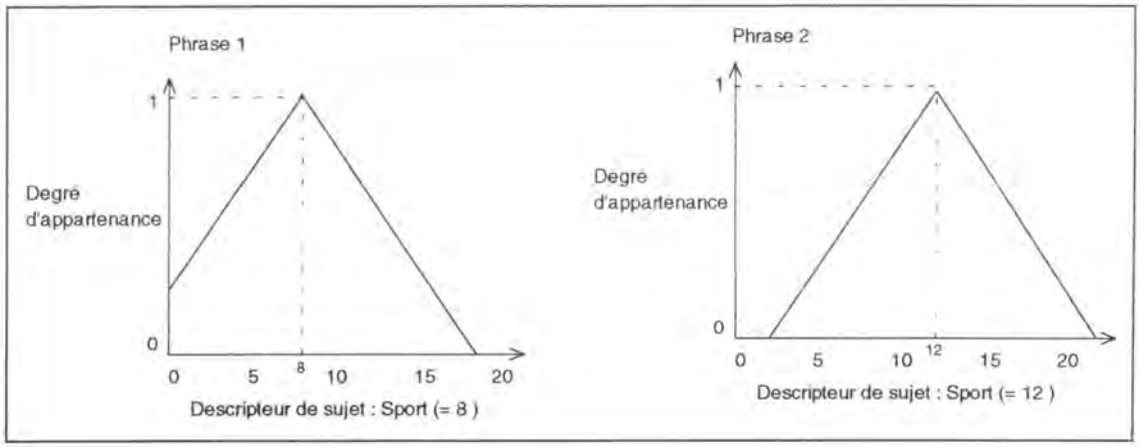

Figure 4.8 : Construction de deux ensembles flous triangulaires

- Pour chacune des deux valeurs (comprises entre 1 et 20), le système construit un ensemble triangulaire flou. Prenons, par exemple, les valeurs 8 et 12 (Figure 4.8).
- Le système modifie ensuite les deux ensembles flous obtenus en tenant compte des fréquences des phrases concernées. Ainsi, si la fréquence de la phrase vaut 1 ou 2, l'ensemble flou concerné sera rendu plus étroit (très étroit dans le cas de 1). De même, si la fréquence de la phrase vaut 4 ou 5, l'ensemble flou concerné sera rendu plus large (très large dans le cas de 5). Si la fréquence de la phrase vaut 3, l'ensemble flou concerné ne sera pas modifié. Pour faire le lien avec la théorie des ensembles flous, ceci peut être vu comme l'application d'un transformateur de courbes sur ces ensembles. Prenons pour notre exemple des fréquences de 4 pour la première phrase et de 1 pour la seconde (Figure 4.9).
- Le système va construire l'intersection de ces deux ensembles flous obtenus et prendre la valeur maximum de l'intersection. On peut faire le lien encore une fois avec la théorie des ensembles flous, en considérant cette étape comme l'application de l'opérateur *intersection* entre deux ensembles. Dans notre exemple, le résultat obtenu est 0.81 (Figure 4.10).

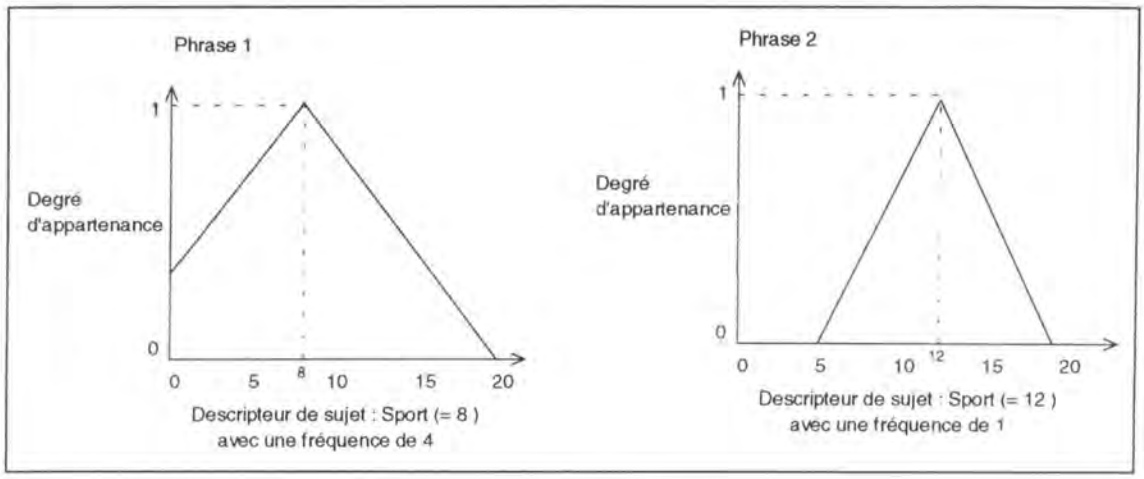

Figure 4.9 : Application de transformateurs sur les ensembles flous

La valeur obtenue sera d'autant plus grande (proche de 1) que les deux valeurs de départ sont proches l'une de l'autre. Grâce à ce système, on permet d'inclure dans cette différence le calcul des fréquences d'apparition de chaque phrase.

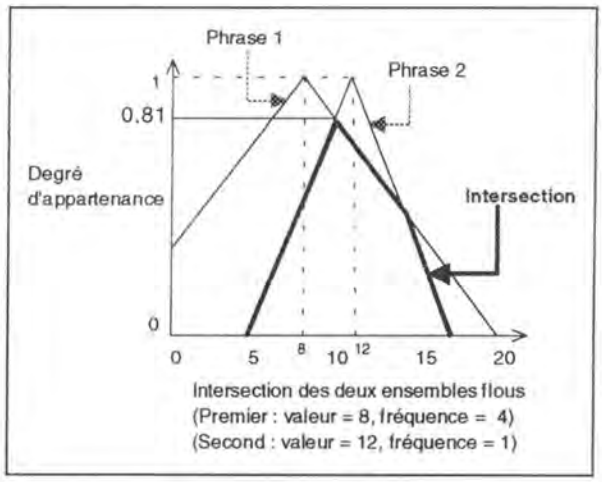

Figure 4.10 : Application de l'opérateur intersection entre les deux ensembles

Par la suite lorsque l'on parle de *comparaison*, c'est dans le sens expliqué dans cette partie. Il y a juste un petit changement en ce qui concerne le calcul de la fréquence de la phrase principale (c'est-à-dire de la phrase qui va servir de référence dans la recherche). En effet, on ne modifie pas la courbe de son ensemble flou avec sa fréquence.

4.5.3. L'algorithme

L'algorithme de prédiction et de recherche d'information utilisant la théorie des ensembles flous est décrit à la Figure 4.11.

A chaque phrase de la base de données est associée une variable *used* initialisée à 10 au lancement du système. Elle permet de ne pas reproposer directement les phrases déjà présentées à l'écran. A chaque nouvelle recherche dans la base de données, la variable *used* de chaque phrase est :

- Soit mise à 0 si la phrase correspondante est présentée à l'écran.
- Soit incrémentée de 1 si la phrase correspondante n'est pas présentée à l'écran (excepté si la valeur de *used* est déjà 10).

```
Au lancement du système :
la variable used de chaque phrase de la base de données = 10

Algorithme flou de prédiction :
Pour les cinq phrases affichées à l'écran, leur variable used = 0
Pour chaque phrase dans la base de données
            (0 ≤ x ≤ Nb de phrase dans la BD  et x <> phrase principale) :
    itemprofile = used de x
    Pour chaque descripteur sujets (0 ≤ y ≤ Nb de descripteurs sujets) :
        itemprofile = itemprofile + la différence entre le descripteur y
        de la phrase principale et le descripteur y de la phrase x.
    Pour chaque descripteur buts(0 ≤ y ≤ Nb de descripteurs buts) :
        itemprofile = itemprofile + la différence entre le descripteur y de
        la phrase principale et le descripteur y de la phrase x.
    Pour chaque descripteur personnes (0 ≤ y ≤ Nb de descripteurs personnes)
        itemprofile = itemprofile + la différence entre le descripteur y de
        la phrase principale et le descripteur y de la phrase x.
    used de x = used de x + 1 (si used < 10)
Prendre les quatre phrases avec le plus grand itemprofile.
```

Figure 4.11 : L'algorithme flou de prédiction

De cette manière, plus une phrase a été récemment proposée à l'écran, plus la variable *used* de cette phrase sera proche de 0.

La variable *itemprofile* est une variable locale à la procédure. Elle contiendra la somme de toutes les différences (au sens de la section précédente) des descripteurs entre deux phrases.

Le principe de l'algorithme flou pour chaque prédiction dans le système est le suivant : Le système doit trouver quatre phrases qui sont les plus proches sémantiquement de la phrase principale (dernière phrase sélectionnée). Il va comparer chacune des phrases de la base de données à la phrase principale en comparant (voir section précédente) leurs différents descripteurs (sujets, buts et personnes).

Le résultat de ces comparaisons sera additionné dans la variable *itemprofile*, qui sera au préalable initialisée par la valeur de la variable *used* de la phrase examinée. Ainsi, les phrases récemment proposées à l'écran durant la conversation seront dès le départ défavorisées. Les quatre phrases choisies seront les phrases qui auront la variable *itemprofile* la plus élevée.

Le système tient compte des données dans le fichier 'diary.in', en faisant une simple multiplication des résultats obtenus lors de la comparaison des descripteurs présents dans ce fichier (nous ne l'avons pas reprise dans la Figure 4.11 pour des raisons de clarté). Cette multiplication donne plus d'importance aux résultats de ces descripteurs par rapport aux autres.

Chapitre 5

Développement de l'interface du système

Dans ce chapitre, nous expliquerons les différentes parties de l'interface du système. Nous avons essayé d'appuyer nos choix sur différents critères ergonomiques et de maintenir autant que possible une cohérence interapplication.

Dans un premier temps, nous expliquerons quels sont les buts de cette interface. Nous poursuivrons en détaillant chaque partie de l'écran principal du logiciel.

Nous mettrons également en évidence la manière dont la souris est utilisée dans le logiciel, ainsi que les différentes formes que peut prendre son curseur.

Nous conclurons ce chapitre en faisant le point sur la rapide évaluation de l'interface, effectuée au cours de sa création.

5.1. Buts de l'interface

Le système FuzzyChat utilise une interface en mode graphique sous DOS (Figure 5.1). Son but est de permettre à l'utilisateur de réaliser le mieux possible la tâche qu'il doit accomplir, avec précision, rapidité et sans effort inutile. Pour reprendre [Bodart, 90], l'objectif de l'ergonomie des interfaces homme-machine peut se résumer en trois points :

- Limiter les baisses de performance lors de l'utilisation d'un nouveau système : baisse de rapidité d'exécution, d'erreurs, multiplication des essais pour atteindre un but.
- Réduire la durée d'apprentissage.
- Limiter la sous-utilisation (exemple : fonctions jamais utilisées dans un traitement de textes, dans un tableur, ...).

L'interface a donc été construite en prenant en compte le plus possible des facteurs humains que l'on a besoin de considérer dans la construction d'un système informatique interactif. Par exemple, le système essaye dans la mesure du possible de donner à l'utilisateur le maximum de *feedback* de ses actions, à partir des couleurs à l'écran, de la souris et du synthétiseur de voix. (*To offer informative feedback* étant une des huit règles d'or dans [Shneiderman, 86])

La construction de l'écran principal essaye aussi de tenir compte de certaines des règles émises par Smith et Mosier [Smith, 84] concernant la présentation des données à l'écran. Voici trois exemples de règles qui ont été prises en considération :

- *Use short and simple sentences;*
- *For any particular type of data display, maintain consistent format from one display to another;*
- *Consider colour coding for application in which users must rapidly distinguish among several categories of data, particularly when the data items are dispersed on the display.*

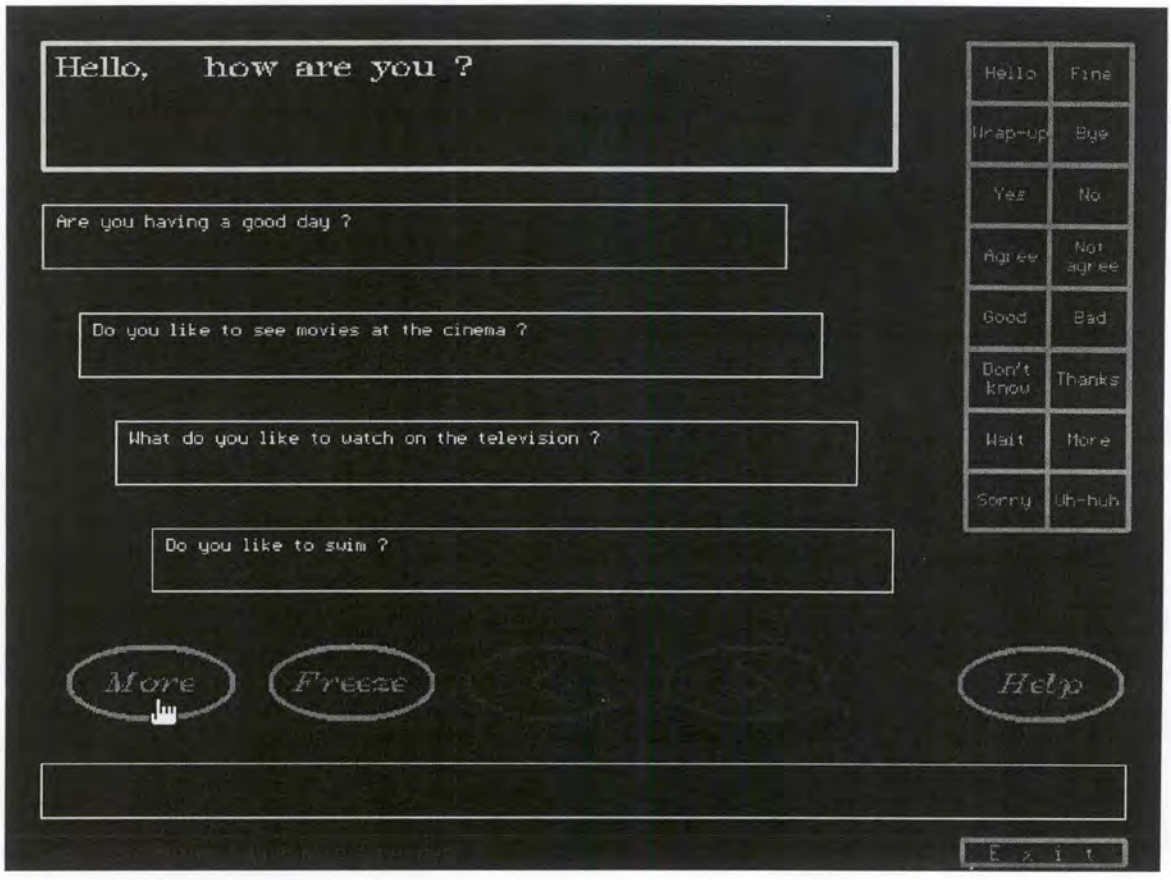

Figure 5.1 : Ecran principal du logiciel FuzzyChat

De même, certaines règles venant du Corpus ergonomique minimal [Vanderdonckt, 92] ont été prises en compte dans la création globale des écrans :

- *Utiliser une même couleur pour grouper des oic[1] liés.*
- *La densité d'affichage d'un écran, d'une boîte de dialogue, d'une fenêtre doit être acceptable.*
- *L'arrangement des objets interactifs d'action doit minimiser le temps de déplacement du curseur vers les actions fondamentales.*

Plus concrètement, cette interface offre trois niveaux possibles de conversation. Au premier niveau se trouve l'espace de prédiction (les cinq boîtes dans le coin supérieur gauche), proposant à l'utilisateur les phrases stockées dans la base de données. Au deuxième niveau, la boîte de dialogue rapide (la boîte dans le coin supérieur droit) permet à l'utilisateur de faire de petits et rapides commentaires courants. Au troisième ni-

[1] Objets Interactifs Concrets

veau se trouve la boîte de création de texte manuel (la boîte située au-dessus des cinq boutons d'option). Elle donne la possibilité de créer du texte à l'aide du clavier. Avec ces trois niveaux de conversation, il est possible de tester le système dans une situation de conversation assez réaliste.

5.2. L'espace de prédiction

L'espace de prédiction est constitué des cinq boîtes situées dans le coin supérieur gauche de l'écran (Figure 5.1). C'est sur ces boîtes que s'applique l'algorithme flou de recherche d'information. La plus grande boîte est la boîte principale, celle contenant la phrase qui vient d'être sélectionnée par l'utilisateur (et donc *parlée* par le synthétiseur de voix). Les quatre autres boîtes sont les quatre phrases (trouvées par l'algorithme flou) proposées à l'utilisateur. Ces quatre phrases sont donc les phrases de la base de données les plus proches sémantiquement de la phrase principale (en tenant compte de leur fréquence et de leur utilisation récente).

En considérant l'écran de la Figure 5.1, l'utilisateur a la possibilité de choisir une des quatre propositions. Supposons qu'il choisisse (c'est-à-dire clique sur cette boîte avec la souris) la phrase '*Do you like to swim ?*', les actions suivantes sont prévues :

- Le synthétiseur de voix va *dire* la phrase.
- Si le système de prédiction n'est pas désactivé par le bouton « FREEZE » :
  - La phrase '*Do you like to swim ?*' va être placée dans la boîte principale.
  - Quatre nouvelles phrases vont être proposées à l'utilisateur. Ces phrases seront sémantiquement proches de la phrase principale.

La taille de la boîte principale est plus large et plus grande que les autres pour indiquer son importance aux yeux des intervenants dans la conversation. La décision d'avoir cinq boîtes dans l'espace de prédiction vient de la recommandation suivante [Shneiderman, 86] : *the number of items to choose from a menu must be 7 +/- 2*. Ainsi l'écran n'est pas surchargé par un nombre trop grand de propositions de phrases.

5.3. Les boutons d'options

Il y a, dans le système, six boutons d'option : « More », « Freeze » / « Unfreeze », « < », « > », « Help » et « Exit ». Ces boutons sont placés au-dessous de l'espace de prédiction et de la boîte de dialogue rapide :

- **Le bouton « *More* »** permet à l'utilisateur de relancer l'algorithme flou de prédiction et de trouver quatre autres phrases.
- **Le bouton « *Freeze* » / « *Unfreeze* »** permet de désactiver ou de réactiver l'espace de prédiction. Lorsque l'espace de prédiction est désactivé, le fait de sélectionner une boîte ne provoque plus une remise à jour de l'espace de prédiction. Seul le synthétiseur de voix fonctionne.
  - Le fait d'appuyer sur « Freeze » (désactivation) change la couleur des cinq boîtes de l'espace de prédiction. Ceci dans le but de donner un bon *feedback* de l'action à l'utilisateur. De plus, le bouton « Freeze » est remplacé par le bouton « Unfreeze », pour que l'action corresponde bien à la tâche que l'utilisateur voudrait exécuter.
  - Le fait d'appuyer sur « Unfreeze » (réactivation) rétablit la couleur des cinq boîtes dans leur couleur initiale et remplace le bouton « Unfreeze » par le bouton « Freeze ». Le système de prédiction est alors réactivé.
- **Les boutons « < » et « > »** sont les boutons de *backtracking*. Ils permettent de se déplacer dans les écrans précédents au cas où l'utilisateur le voudrait. Ces deux boutons peuvent être inactifs à certains moments (lorsqu'on est au premier écran, dans le cas du bouton « < » ou lorsqu'on est au dernier écran, dans le cas du bouton « > »). Leur inactivité se reflète par un changement de couleur (plus foncé). Ainsi, la règle suivante dans [Vanderdonckt, 92] est respectée : *Les présentations des états non sélectionnés, sélectionnés et inactifs d'une boîte doivent être suffisamment distinctes.*
- **Le bouton « *Help* »** permet aux nouveaux utilisateurs d'obtenir deux écrans d'aide (Figure 5.2 et Figure 5.3) expliquant le principe de fonctionnement du logiciel, ainsi que ses diverses fonctions.
- **Le bouton « *Exit* »** permet à l'utilisateur de finir sa session FuzzyChat.

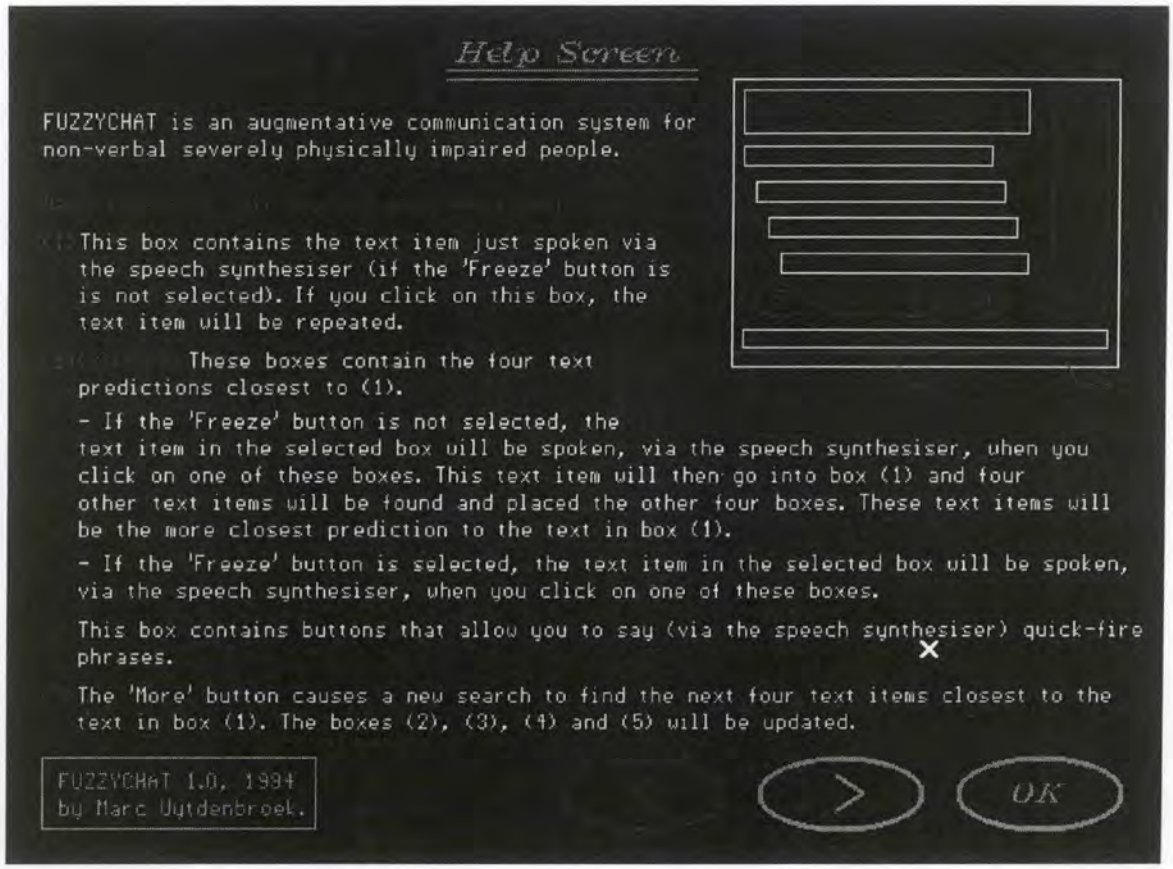

Figure 5.2 : Premier écran d'aide

Ces boutons ont été conçus en respectant les règles suivantes ([Vanderdonckt, 92]) :

- *Les dénominations des données affichées et de leurs libellés identificatifs doivent incorporer les termes familiers, clairs ou propres à la tâche de l'utilisateur et éviter les termes techniques appartenant à l'informatique.*
- *Seule l'initiale de tout libellé doit être en majuscule.*
- *Si une boîte de dialogue ne présente aucun bouton par défaut, aucun bouton ne doit avoir de présentation spéciale.*
- *Tous les boutons de commande d'un même groupe doivent être équidistants l'un de l'autre.*
- *Les boutons de commande, de navigation, de déplacement doivent être positionnés de manière logique et cohérente* (« < » et « > »).

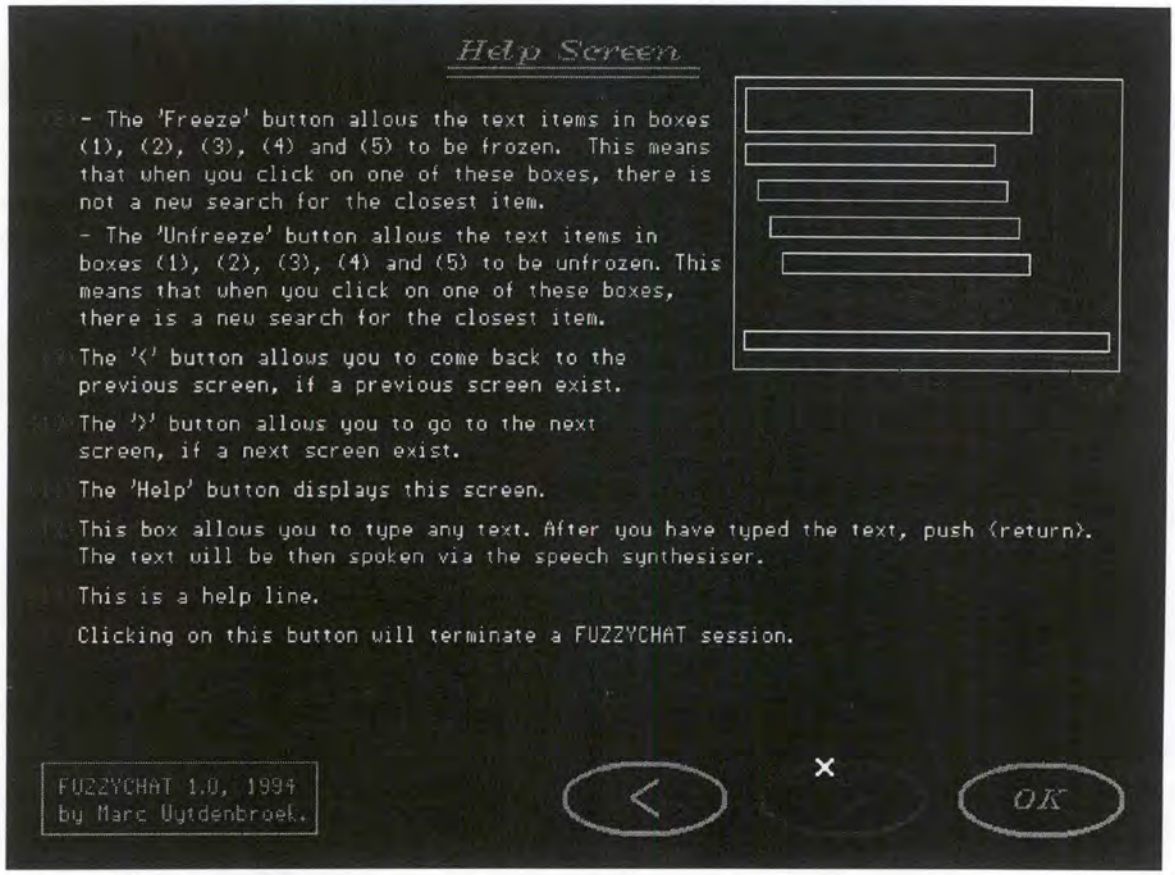

Figure 5.3 : Second écran d'aide

- *Les boutons de commande disposés horizontalement doivent être équilibrés verticalement.*
- *Le nombre de boutons de commande, de boutons graphiques ne peut excéder 7 par oic*[1] *composé.*

De même, pour donner un maximum de *feedback* à l'utilisateur, lorsque celui-ci presse sur un des ces boutons avec la souris, le bouton en question change de couleur (plus foncé), le temps que l'utilisateur le garde appuyé.

5.4. La boîte de dialogue rapide

La boîte de dialogue rapide se trouve dans le coin supérieur droit de l'écran (Figure 5.1). Elle correspond à une version simplifiée du système CHAT [Newel, 92], développé au

---

[1] Objet Interactif Concret.

MicroCentre de l'université de Dundee. Cette boîte de dialogue permet à l'utilisateur de *dire* (par le synthétiseur de voix) de petites phrases qui reviennent souvent dans une conversation. Cela aide la personne à prendre pleinement part à une communication.

De même certaines règles ergonomiques, citées plus haut, ont été appliquées pour obtenir une interface mieux adaptée aux actions des utilisateurs.

5.5. Création de texte et aide en ligne

L'utilisateur peut créer manuellement du texte en cliquant directement avec la souris dans la boîte située au-dessous des boutons d'options (Figure 5.1). Le pointeur de la souris disparaît alors pour laisser place à un curseur situé dans la boîte de texte. L'utilisateur tape son texte et appuie sur <return> pour que, ce qu'il a tapé, soit *parlé* via le synthétiseur de voix. Le pointeur de la souris refait alors son apparition. Encore une fois, le respect de la règle suivante dans [Vanderdonckt, 92] a été appliqué : *L'invitation à saisir des menus pleins écran doit être un symbole spécial réservé à cet effet.*

Il existe, pour finir, dans la partie inférieure de l'écran (Figure 5.1) une aide en ligne. Cela permet aux utilisateurs peu expérimentés d'être guidés tout au long de l'utilisation du logiciel, sans devoir passer chaque fois par les écrans d'aide. L'existence de cette ligne d'aide est due aux deux règles suivantes ([Vanderdonckt, 92]) :

- *Réserver plusieurs lignes parmi les dernières lignes de l'écran pour afficher les messages d'état, d'erreur.*
- *Les messages de guidage et d'erreur, relatifs à différentes fenêtres doivent être affichés à la base de chaque fenêtre.*

5.6. Utilisation de la souris

Le curseur de la souris joue un rôle important dans le système. Il change son apparence en fonction de sa place à l'écran (Figure 5.4). Cela permet à l'utilisateur de visualiser les différentes actions qu'aurait un cliquement de la souris dans une région bien particulière de l'écran. Seul le bouton de gauche de la souris est actif.

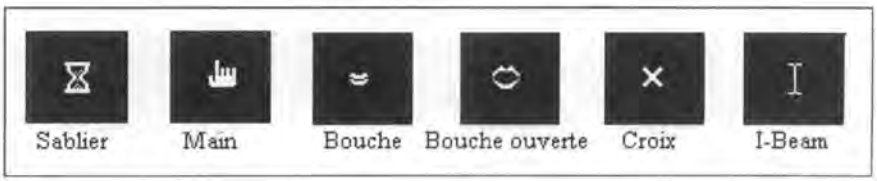

Figure 5.4 : Les différentes formes que peut prendre le curseur de la souris

Le curseur de la souris peut prendre les formes suivantes :

- Le *sablier* signifie que le système est en état de recherche et que l'utilisateur doit attendre. Quand l'algorithme flou s'exécute, le *sablier* apparaît et toutes les fonctions du logiciel sont désactivées.
- Lorsque le curseur de la souris se trouve à l'intérieur d'un des boutons d'option, il a l'apparence d'une *main*, avec un doigt pointé en avant. L'utilisateur aura ainsi l'impression d'appuyer sur ces boutons avec le doigt.
- Tant que le curseur est à l'intérieur d'une des cinq boîtes de prédiction ou à l'intérieur de la boîte de dialogue rapide, il a l'apparence d'une *bouche*. Ceci pour indiquer qu'une sortie sonore parlée sera émise si l'utilisateur appuie à ce moment-là. Dans ce dernier cas, la *bouche ouverte* apparaît un bref instant, confirmant aux yeux de l'utilisateur qu'une action a été exécutée.
- La croix signifie qu'il n'y a pas d'action exécutée par le système si l'utilisateur presse le bouton de la souris.
- Lorsque le curseur se trouve dans la boîte de texte manuel, il a l'apparence d'un *I-beam*.

Le choix des curseurs de la souris a été fait pour essayer de respecter au maximum la cohérence interapplication (Microsoft Windows, X-Windows, ...).

5.7. Evaluation en cours de construction de l'interface

Durant la création de celle-ci, une évaluation subjective de l'interface fut réalisée, afin de récolter des avis et des conseils de personnes expérimentées dans le domaine des prothèses de communication. Cette évaluation fut réalisée auprès d'une dizaine de personnes ayant des statuts différents : étudiants, chercheurs et professeurs.

L'évaluation de l'interface du logiciel a été faite de la façon suivante : après une utilisation et une visualisation du logiciel par l'examinateur, celui-ci donna son avis (points positifs et négatifs) au sujet de l'interface existante. L'examinateur décrit également les modifications et les additions que le logiciel devrait subir selon lui.

Quelques commentaires qui ont contribué à la réalisation de l'interface finale

« *Not sure about the relevance of the 'text' button. Why not just have the text box on the screen all the time ready for use* » H. Glyn.

« *Include a help button to describe more about the system* » A. Al-Taani.

« Add a help window and an exit button » N. Devnani.

« *Put a help line facility* » F. Moisy.

Suite à cette évaluation, l'interface fut en partie modifiée. Tous les commentaires et avis n'ont pu être pris en compte, soit par manque de temps, soit parce qu'ils se contredisaient. Néanmoins, les conséquences de cette évaluation sont l'ajout de trois spécifications (voir *4.1.2. Spécification des exigences*) dans le système :

- Création d'un bouton « Exit ».
- Création d'un bouton « Help ».
- Création d'une ligne d'aide.

Une évaluation globale du logiciel a également été faite sur la version finale du logiciel. Cette évaluation tient compte de tous les aspects du système et notamment de l'interface. Le septième chapitre en traite plus longuement.

Chapitre 6

Détails de l'implémentation

Dans ce chapitre, nous jetons un oeil sur les différents détails techniques du logiciel. Nous essayons de rester dans un cadre assez général de peur que le lecteur ne se perde dans des détails de programmation.

Après une brève description des éléments techniques généraux utilisés dans le système, nous décrirons comment celui-ci a été divisé en trois sous-systèmes utiles. La description des différents modules et fonctionnalités du logiciel sera également réalisée.

6.1. En général

Voici les détails techniques généraux propres à la réalisation du logiciel FuzzyChat :

- Il s'exécute sur PC IBM ou compatibles.
- Il est construit pour fonctionner avec un synthétiseur de voix : *Dolphin Apollo Speech Synthesiser*.
- Il a été écrit en Borland C++ 3.1. sous DOS.
- Il utilise une interface en mode graphique.
- Il a été construit en tenant compte d'une approche orienté-objets.

FuzzyChat fut réalisé au MicroCentre (centre de recherche appartenant au département mathématique et informatique) de l'université de Dundee (Ecosse), avec la collaboration de Marianne Hickey (*Lecturer in Computer Science*) et de Norman Alm (*Doctor in Computer Science*).

Le lecteur, qui veut en connaître davantage sur les détails de l'implémentation du système, trouvera en Annexe A, le listing complet et commenté de l'ensemble du logiciel.

6.2. Les modules

La construction du système FuzzyChat a été divisée en trois sous-systèmes (Figure 6.1), ceci dans le but d'avoir une première, puis une seconde version exécutable, dans le cas où le temps manquerait pour achever l'ensemble du système.

Le premier module du système est **le module *Coordinateur*** (*coordinating*). Il contient la déclaration de tous les fichiers *headers*, de toutes les constantes et de toutes les variables globales du système. Il contient également la fonction principale *(main())* du logiciel FuzzyChat. Comme son nom l'indique, c'est ce module qui va coordonner l'ensemble du système. Il se trouve dans le fichier 'coordina.cpp'.

Les fichiers 'litt.chr', trip.chr' et 'tscr.chr' sont les fichiers qui définissent les polices de caractères utilisées dans le système. De même, le fichier 'egavga.bgi' permet d'utiliser le mode graphique du DOS.

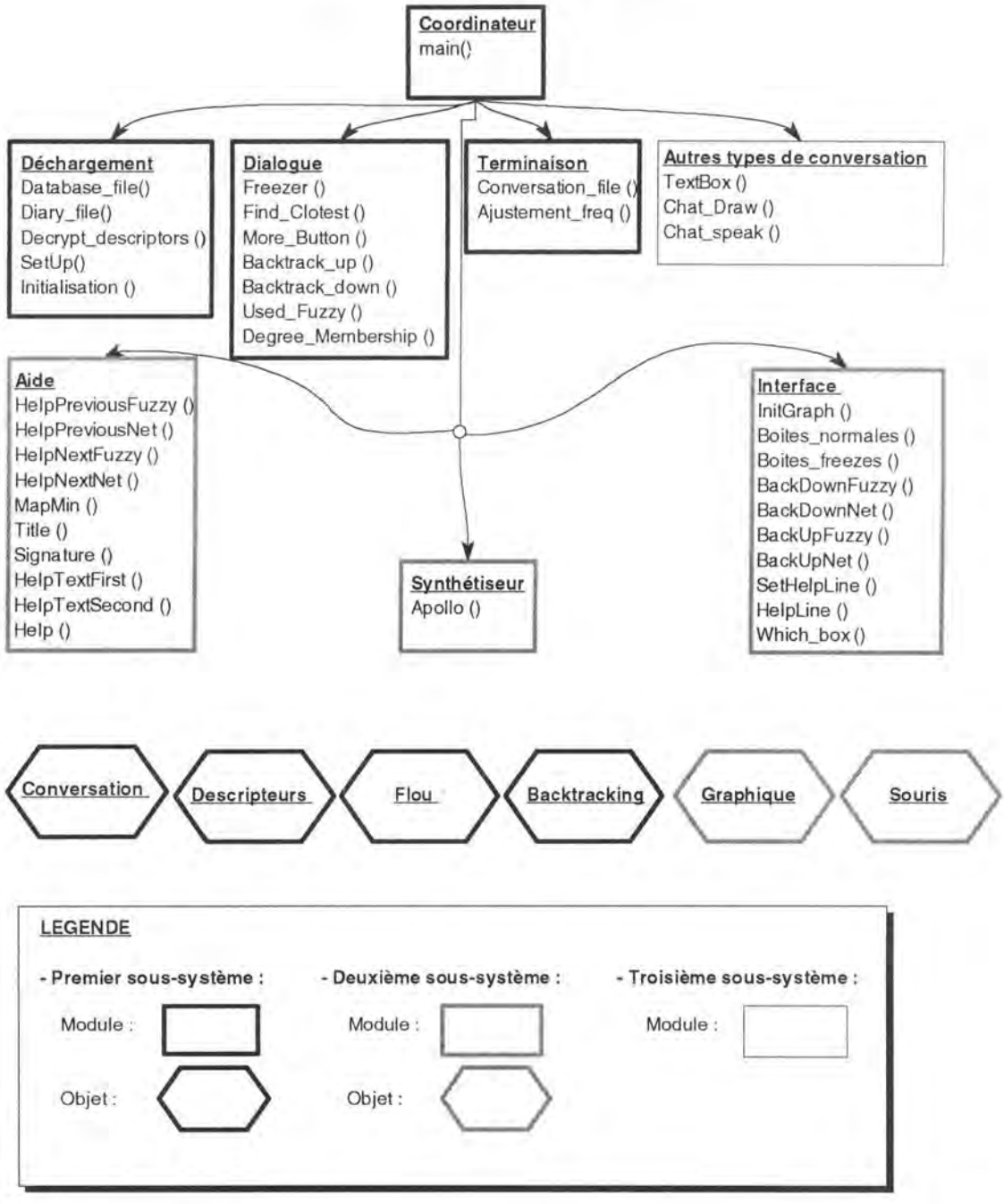

Figure 6.1 : Les modules de FuzzyChat et leurs fonctions

6.2.1. Le premier sous-système

Le premier sous-système est principalement constitué par le système de prédiction. C'est-à-dire qu'il inclut toutes les fonctionnalités nécessaires à la recherche d'information dans la base de données. Il comprend notamment l'algorithme flou de prédiction. Ce premier sous-système fonctionne avec le minimum d'interface homme-machine, c'est-à-

dire qu'il s'occupe de fournir à l'écran les résultats de ses recherches, seulement de façon brute.

Il inclut les modules suivants (bien sûr, ces modules sont maintenant dans leur version finale, c'est-à-dire qu'ils comprennent les appels aux modules des deuxième et troisième sous-systèmes) :

- **Le module *Déchargement*** (fichier 'Unloadin.cpp') : ce module regroupe les fonctions de déchargement du logiciel création de la base de données et initialisation des variables globales nécessaires au lancement du système. Il reprend les fonctions suivantes :
  - *Database_file* : décharge le fichier 'Database.in'.
  - *Diary_file.* : décharge le fichier 'Diary.in'.
  - *Decrypt_descriptors* : ajuste le poids des descripteurs selon les données du fichier 'Diary.in'.
  - *SetUp* : décharge le fichier 'Setup.in'.
  - *Initialisation* : initialise les variables globales et lance une première fois le système de prédiction.
- **Le module *Dialogue*** (fichier 'Speechbox.cpp') : Ce module regroupe l'ensemble des fonctions de dialogue du logiciel, c'est-à-dire l'espace de prédiction et les boutons d'option « More », « Freeze » / « Unfreeze », « < » et « > ». C'est dans ce module que se trouve l'algorithme flou de prédiction. Il reprend les fonctionnalités suivantes :
  - *Freezer* : traite l'option « Freeze » / « Unfreeze ».
  - *Find_Clotest* : traite l'espace de prédiction.
  - *More_Button* : traite l'option « More ».
  - *Backtrack_up* : traite l'option « > ».
  - *Backtrack_down* : traite l'option « < ».
  - *Used_Fuzzy* : analyse la base de données pour trouver les quatre phrases à prédire, en fonction de la phrase courante.
  - *Degree_Membership* : calcule la *différence floue* entre les descripteurs de deux phrases (Algorithme flou).

- **Le module *Terminaison*** (fichier 'Finish.cpp') : ce module comprend les fonctions de terminaison du système. Il reprend les fonctions suivantes :
    - *Conversation_file* : crée le fichier 'Conversa.out'.
    - *Ajustement_freq* : met à jour la fréquence des phrases dans le fichier 'Database.in'.

Le premier sous-système inclut également la création de 4 objets :

- **L'objet *Backtracking*** (fichier 'Backtrac.cpp') : cet objet s'occupe de la gestion du système de *backtracking* du logiciel.
- **L'objet *Conversation*** (fichier 'Conversa.cpp') : cet objet regroupe la base de données (avec l'objet *Descripteurs*) du système. Il contient l'ensemble des phrases de l'utilisateur (venant du fichier 'Database.in') et chacune des valeurs des descripteurs.
- **L'objet *Descripteurs*** (fichier 'Descript.cpp') : il regroupe la base de données (avec l'objet *Conversation*) du système. Il contient le nom des différents descripteurs, ainsi que leur poids (venant du fichier 'Diary.in'), dont le système tiendra compte lors de la recherche d'information.
- **L'objet *Flou*** (fichier 'Fzy_unit.cpp') : il contient les fonctionnalités floues de base permettant de créer un ensemble triangulaire flou et de trouver l'intersection de deux ensembles flous. Il fait également appel à un fichier ('data_fzy') lui définissant certaines constantes.

6.2.2. Le deuxième sous-système

Le deuxième sous-système est constitué du premier sous-système auquel on a rajouté les fonctions nécessaires pour obtenir une interface homme-machine adaptée. Il inclut également l'utilisation de la souris et du synthétiseur de voix. A cette étape, le projet pouvait déjà faire l'objet de tests concrets auprès d'utilisateurs.

Il inclut les modules suivants (bien sûr, ces modules sont maintenant dans leur version finale, c'est-à-dire qu'ils comprennent les appels au module du troisième sous-système) :

- **Le module *Interface*** (fichier 'graphic.cpp') : il regroupe l'ensemble des fonctions de gestion de l'interface homme-machine du logiciel. Il reprend les fonctionnalités suivantes :

- *InitGraph* : initialise le mode graphique du système.
- *Boites_normales* : dessine, en mode *unfreeze*, les cinq boîtes de l'espace de prédiction.
- *Boites_freezes* : dessine, en mode *freeze*, les cinq boîtes de l'espace de prédiction.
- *BackDownFuzzy* : dessine, en mode inactif, le bouton « < ».
- *BackDownNet* : dessine, en mode actif, le bouton « < ».
- *BackUpFuzzy* : dessine, en mode inactif, le bouton « > ».
- *BackUpNet* : dessine, en mode actif, le bouton « > ».
- *SetHelpLine* : dessine la ligne d'aide (dernière ligne de l'écran).
- *HelpLine* : met à jour la ligne d'aide du système.
- *Which_box* : met à jour le curseur de la souris et détermine la position de la souris (lorsque le bouton de celle-ci a été pressé).

- **Le module *Aide*** (fichier 'help.cpp') : il gère les deux écrans d'aide du système. Il contient les fonctions suivantes :
  - *HelpPreviousFuzzy* : dessine, en mode inactif, le bouton « < » de l'écran d'aide.
  - *HelpPreviousNet* : dessine, en mode actif, le bouton « < » de l'écran d'aide.
  - *HelpNextFuzzy* : dessine, en mode inactif, le bouton « > » de l'écran d'aide.
  - *HelpNextNet* : dessine, en mode actif, le bouton « > » de l'écran d'aide.
  - *MapMin* : dessine l'interface du système de façon réduite.
  - *Title* : dessine le titre de l'écran d'aide.
  - *Signature* : dessine la signature de l'écran d'aide.
  - *HelpTextFirst* : écrit le texte du premier écran d'aide.
  - *HelpTextSecond* : écrit le texte du second écran d'aide.
  - *Help* : gère les actions de la souris dans les écrans d'aide.

- **Le module *Synthétiseur*** (fichier 'synth.cpp') : il gère la fonction *appollo* qui permet de faire fonctionner le synthétiseur de voix.

Le deuxième sous-système inclut également la création de deux objets supplémentaires :

- **L'objet *Graphique*** (fichier 'graphobj.cpp') : définit les objets *rectangle* et *button*. Il permet la gestion des boutons et des rectangles du système.
- **L'objet *Souris*** (fichiers 'msmouse.cpp' et 'msmouse.hpp') : cet objet permet de gérer l'utilisation de la souris tout au long de l'exécution du système. Il fait appel au fichier 'mcursor.cpp' pour la création des différentes formes que peut prendre le curseur de la souris à l'écran.

6.2.3. Le troisième sous-système

Le troisième sous-système, qui est le système final FuzzyChat, ajoute aux deux sous-systèmes précédents la possibilité de faire des commentaires rapides et répétitifs, ainsi que de permettre à l'utilisateur de créer manuellement du texte. Ceci dans le but d'avoir un système de communication le plus complet possible.

Il inclut donc au reste du système **le module *Autres types de conversation*** (fichier 'textchat.cpp'). Il contient les fonctions suivantes :

- *TextBox* : gère la possibilité de créer manuellement du texte.
- *Chat_Draw* : dessine la boîte permettant de faire de rapides et répétitifs commentaires.
- *Chat_speak* : gère l'utilisation de la boîte de commentaires rapides et répétitifs.

Chapitre 7

Evaluation du système et travaux futurs

Dans ce chapitre, nous faisons un premier bilan de l'ensemble du projet FuzzyChat, tout en sachant que d'autres enquêtes doivent encore être réalisées pour valider ce type de système d'aide à la communication.

Dans une première partie, nous décrirons l'enquête qui a été réalisée en fin de projet. Cette enquête avait pour objet d'évaluer les différentes parties du logiciel, comprenant une évaluation de la conversation, de l'outil, de l'interface et du synthétiseur de voix.

Dans la seconde partie de ce chapitre, nous élaborerons quelques pistes de recherche, permettant d'améliorer le projet en cours.

7.1. Evaluation du système

7.1.1. L'enquête

En fin de projet, une enquête fut réalisée pour évaluer les performances du système FuzzyChat et pour donner une réponse au principal objectif de ce projet, à savoir explorer la possibilité d'appliquer la théorie des ensembles flous à un système d'aide à la communication pour les personnes souffrant de problèmes de langage.

Cette enquête fut divisée en deux parties :

- La première partie avait comme but de recréer au maximum une conversation naturelle à partir du système FuzzyChat : une personne —à l'aide du synthétiseur de voix— utilisait le logiciel pour converser avec une autre personne qui, elle, parlait normalement. Cette conversation dura entre 10 et 15 minutes à chaque fois.
- Dans la seconde partie de cette enquête, la personne, ayant parlé normalement durant la conversation, répondait à un petit questionnaire. Ce questionnaire (Figure 7.1 à la Figure 7.4) s'appuyait sur différents critères :
    - La qualité de la conversation.
    - Les améliorations et changements que l'on pourrait apporter au système.
    - L'interface homme-machine.
    - La qualité du synthétiseur de voix.

Cette enquête a été réalisée auprès de 8 personnes. Ces personnes n'étaient pas en contact avec le logiciel, elles se contentaient de dialoguer avec la personne qui utilisait le système FuzzyChat. Cette personne était toujours le développeur du logiciel. En effet, les fichiers d'entrée, ('Database.in' et 'Diary.in') étant strictement personnels, un travail d'encodage préliminaire à l'utilisation du système doit être fait pour chaque personne qui veut l'utiliser. Ce travail étant long et fastidieux, il était difficile de demander à ces 8 personnes de créer leur propre fichier 'Database.in' et 'Diary.in'.

Il est donc nécessaire de se rappeler que cette évaluation est, en partie, biaisée. En effet, tout l'aspect utilisation (interface homme-machine) et compréhension du système devront encore être testés, auprès de personnes souffrant de problèmes de langage.

- Ainsi, il est nécessaire de vérifier si la phase préliminaire d'encodage des deux fichiers personnels d'entrée ('Database.in' et 'Diary.in') n'est pas trop lourde pour ce type de personne. Une aide extérieure (famille ou autres) devra sans doute être requise pour encoder l'ensemble des phrases que l'utilisateur voudra dire. Cette étape est assez lourde mais ne doit être réalisée qu'une seule fois. En effet, une fois que la base de données est entrée, seuls de petits changements ou modifications devront être encore faits régulièrement.
- De même, il serait intéressant d'évaluer la capacité de la personne handicapée à assigner des valeurs (entre 1 et 20) pour les différents descripteurs des phrases de sa base de données. En effet, il est possible que certaines personnes aient des difficultés à situer une phrase dans une échelle allant de 1 à 20.
- La manipulation du logiciel pour une personne *mal-parlante* devra également être évaluée. En effet, ces personnes peuvent aussi souffrir de problèmes de coordination dans leurs mouvements. Il serait notamment intéressant de savoir si l'usage de la souris constitue le meilleur moyen d'utiliser le logiciel.
- Pour finir, la compréhension du logiciel par la personne handicapée devra aussi être testée. Puisque c'est le développeur du système qui a utilisé le logiciel durant l'enquête, il est évident que la compréhension des différentes fonctions du logiciel ne lui a posé aucun problème. Il serait intéressant de savoir si, pour d'autres personnes (notamment les personnes ciblées par ce projet), la compréhension se ferait dans d'aussi bonnes conditions.

L'ensemble de ces évaluations devrait être exécuté durant l'année 1995-1996 par le MicroCentre de l'université de Dundee, qui continue la réalisation de ce projet.

Pour revenir à l'enquête proprement dite, elle a été réalisée par des personnes de l'université de Dundee (Ecosse), incluant des professeurs, des chercheurs et des étudiants. Ces personnes ont des formations différentes : ingénieurs informaticiens, ingénieurs électroniciens, ingénieurs physiciens et psychologues. Elles proviennent de plusieurs catégories d'âge (entre approximativement 20 et 50 ans) et sont des deux sexes.

7.1.2. A propos de la qualité de la conversation

L'évaluation de la qualité de la conversation (Figure 7.1) permet de nous donner une idée sur la manière dont les intervenants ont conversé pendant ces 10 à 15 minutes. Elle permet également d'étudier la vitesse de la conversation en général et la rapidité de l'utilisateur à trouver 'les bonnes phrases'.

How did you find the conversation ?

- What went well ?
- What went wrong ?
- What do you think about the pace of the conversation ?
- Did the conversation give you any impression of the personality of the user ?
- Ohter comments about the conversation ?

Figure 7.1 : Evaluation de la conversation (1ère partie du questionnaire)

Les résultats concernant la première partie de cette enquête furent les suivants :

- Généralement, les personnes étaient heureuses de leur conversation. Celle-ci semblait réaliste et correspondait à des conversations de tous les jours.
- Un problème est apparu lorsque l'utilisateur devait utiliser la boîte de création manuelle de texte. Cela ralentissait considérablement la vitesse de conversation. Ainsi lorsque le partenaire posait une question spécifique, la réponse à cette question était lente, si elle ne se trouvait pas dans la base de données.
- Excepté le problème du point précédent, la vitesse de conversation était tout à fait raisonnable.
- La conversation donna un bon aperçu de la personnalité de l'utilisateur. Ceci grâce en partie à sa base de données personnelles.
- La possibilité de dire des petits commentaires rapides et répétitifs dans la conversation était très utile et rendait celle-ci assez naturelle.
- Le système était fort adapté lorsque c'est l'utilisateur qui guidait la conversation, qui posait des questions. Il l'était moins dans le cas contraire, car l'utilisateur devait utiliser plus souvent la boîte de création manuelle de texte.

Avec une base de données conséquente, le système fournit une conversation solide et pas frustrante du tout. Néanmoins, un gros point noir réside dans l'utilisation de la boîte de création manuelle de texte, car elle ralentit considérablement la vitesse de conversation.

Voici quelques-uns des commentaires intéressants apparus lors de la première partie de cette enquête :

- *« The pace of the conversation was fine and not frustrating at all. »*
- *« I felt that I was responding far more than driving the conversation. »*
- *« Couldn't go into any depth about a topic unless you used the keyboard, potentially a problem for some users. »*
- *« What went well ? Topic changing finding out about the person using it. »*
- *« The conversation seemed to be quite realistic. »*

7.1.3. A propos des améliorations et changements à apporter

La deuxième partie du questionnaire demande aux intervenants les différents changements ou améliorations que l'on pourrait apporter au système pour le rendre plus efficient (Figure 7.2). Le résultat se réduit principalement à deux éléments. Le premier concerne l'utilisation des commentaires rapides et répétitifs, le second concerne la boîte de création manuelle de texte.

How could we improve to the system ?

- Additions ?
- Changes ?
- Other comments ?

Figure 7.2 : Changements et améliorations du système (2ème partie du questionnaire)

Selon eux, le système devrait inclure une plus grande liste de petits commentaires rapides et répétitifs. Cela permettrait d'avoir une meilleure conversation : plus authentique et avec plus de 'répondant'. Ainsi, la possibilité de dire 'OK' dans la boîte de commentaires rapides et répétitifs aurait, selon certains, été appropriée à plusieurs moments de la conversation.

Comme on l'a remarqué dans la section précédente, la possibilité de créer manuellement du texte pose un problème. Une des solutions serait d'introduire dans le logiciel un système de prédiction de mots comme celui détaillé dans la section *3.1. Word Prediction : PAL*. Il permettrait d'accélérer la vitesse de frappe de l'utilisateur et donc la vitesse de conversation en général. Une autre solution serait d'augmenter le nombre de phrases présentes dans la base de données. Ainsi, on minimiserait l'utilisation de cette possibilité puisque plus de sujets seraient couverts par le système de prédiction.

7.1.4. A propos de l'interface

L'interface du logiciel est un élément important dans les systèmes de communication de ce type. C'est principalement cette partie qui devra être utilisée, puis évaluée par des personnes souffrant de problèmes de langage, pour que l'on puisse en savoir plus sur la qualité de l'interface.

What do you think about the interface ?

- Easy to understand ?
  - Very easy O
  - Easy O
  - Medium O
  - Difficult O
  - Very Difficult O
- Easy to use ?
  - Very easy O
  - Easy O
  - Medium O
  - Difficult O
  - Very Difficult O
- Other comments about the interface ?

Figure 7.3 : Evaluation de l'interface (3ème partie du questionnaire)

Néanmoins, les résultats obtenus dans cette enquête sont encourageants et fort concluants. Les voici en détail :

- **Point de vue de la compréhensibilité de l'interface :** 37, 5 % des personnes interrogées considèrent la compréhension de l'interface du système FuzzyChat comme *very easy*, 50 % comme *easy* et seulement 12,5% comme *medium*.

- **Point de vue de la facilité à utiliser l'interface :** 37,5 % des personnes interrogées considèrent la facilité à utiliser l'interface du système comme *very easy*, 37,5 % comme *easy* et 25 % comme *medium*.

Tant le respect d'un maximum de règles ergonomiques (voir le chapitre 5), que l'utilisation efficace de la souris dans la construction de l'interface homme-machine, ont permis d'obtenir ces premiers résultats positifs. L'aide en ligne (située à la ligne inférieure de l'écran) et les deux écrans d'aide permettent un guidage efficient des nouveaux utilisateurs tout au long de l'exécution du système. De même, la manière dont le système a été conçu, en essayant d'avoir un maximum de cohérences interapplications, rend l'utilisation de FuzzyChat facile et rapide.

En se référant aux trois objectifs de l'ergonomie des interfaces —à savoir limiter les baisses de performance lors de l'utilisation d'un nouveau système, réduire la durée d'apprentissage et limiter la sous-utilisation (exprimés dans la section *5.1. Buts de l'interface*)— cette première évaluation ne peut être considérée que comme positive. Il est certes intéressant d'attendre les résultats des évaluations faites avec les personnes susceptibles d'utiliser ce logiciel pour en savoir plus.

7.1.5. A propos du synthétiseur de voix

Le but de la dernière partie de ce questionnaire (Figure 7.4) n'est pas d'évaluer la qualité du synthétiseur de voix pour lui-même, mais plutôt d'évaluer le synthétiseur de voix pour la qualité globale du système. En effet, une mauvaise restitution du langage parlé par le synthétiseur pourrait avoir des conséquences sur la qualité et la vitesse de la conversation.

Le synthétiseur *Dolphin Apollo Speech Synthesiser* s'est révélé en général d'un très bon choix, tant du point du vue de sa compréhension que de sa qualité de voix; même s'il pourrait être plus approprié à la personne qui utilise le logiciel (ainsi si l'utilisateur est du sexe féminin, la possibilité d'avoir une voix féminine aurait été un plus dans le système). Voici les résultats détaillés obtenus :

- **Point de vue de la compréhension :** 37,5 % des personnes interrogées considèrent la compréhension du synthétiseur comme *very good*, 50 % comme *good* et seulement 12,5 % comme *medium*.

- **Point de vue de la qualité de voix** (est-elle appropriée à la conversation ?) : 12,5 % des personnes interrogées considèrent la qualité de la voix du synthétiseur comme *very good*, 25 % comme *good*, 50 % comme *medium* et 12,5 % comme *bad*.

What do you think about the speech synthesiser ?

- Understandibility ?
  - Very good O
  - Good O
  - Medium O
  - Bad O
  - Very bad O
- Appropriate voice quality ?
  - Very good O
  - Good O
  - Medium O
  - Bad O
  - Very bad O
- Other comments about the speech synthesiser ?

Figure 7.4 : Evaluation du synthétiseur de voix (4ème partie du questionnaire)

7.1.6. Pour résumer

Jusqu'à présent, on peut dire que ce projet a confirmé le fait qu'un système d'aide à la communication, utilisant la théorie des ensembles flous dans sa recherche d'information (et dans son système de prédiction), peut être utile aux personnes souffrant de problèmes de langage. Il confirme également que le stockage au préalable des données conversationnelles (les phrases) est, dans un système de prédiction, fort commode dans la construction rapide et efficace de conversations.

Cependant, un des grands problèmes du logiciel se situe dans l'option de création manuelle de texte. En effet, dans un système pareil, il est impossible de stocker dans la base de données toutes les réponses aux éventuelles questions de l'interlocuteur. L'utilisateur doit donc passer par cette option pour lui répondre. Ce qui ralentit considérablement la moyenne de la vitesse de conversation. Comme on l'a déjà dit, l'utilisation d'un système de prédiction de mots ([Newell, 91] et [Morris, 92]) pourrait apporter une solution acceptable à ce problème.

Il est important de souligner que cette évaluation nous a renseignés sur le type de conversation offert par le système : ce logiciel serait davantage destiné à guider et mener la conversation, qu'à la suivre. En effet, c'est l'utilisateur de ce système qui doit diriger la discussion avec les phrases qu'il possède dans sa base de données, pour éviter au maximum l'utilisation de l'option de création manuelle de texte.

La rapidité de la conversation est un point positif au système. Restant encore un peu au-dessous de la vitesse d'une discussion normale, elle est tout à fait acceptable et ne provoque aucune frustration auprès des intervenants.

L'interface permet d'utiliser rapidement et facilement le système. Même si des évaluations plus adaptées doivent être réalisées, elle est jusqu'ici un point fort du logiciel, essayant notamment de donner un bon *feedback* à l'utilisateur.

Reprenons, pour conclure, les quatre buts (voir *1.2.2. Buts d'un système d'aide à la communication*) d'un système d'aide à la communication et voyons de quelle manière le système les a atteints :

- **L'expression des besoins** : c'est peut-être ce but qui est le moins bien atteint par le système FuzzyChat. En effet, même si l'on peut stocker dans la base de données l'expression des différents besoins que pourrait avoir l'utilisateur, il est difficile pour un tel système de les reproposer au moment opportun.
- **Le transfert d'information** : c'est sans aucun doute le but le mieux atteint par le système. En effet, l'utilisateur peut, via le logiciel et sa base de données personnelles, transmettre toute l'information qu'il veut, et ceci de façon structurée et ordonnée.
- **Les relations sociales** : pour autant que la base de données soit mise à jour assez régulièrement (pour éviter l'aspect répétitif dans les différentes conversations), ce but est relativement bien atteint par le système.
- **L'étiquette sociale** : même si ce logiciel n'avait pas pour objectif d'atteindre ce but, il pourrait très bien y répondre, pourvu que la base de données y soit adaptée.

7.2. Les travaux futurs

Dans cette section, nous allons donner quelques pistes de recherche pour prolonger et améliorer le projet décrit dans ce travail. La réalisation de chacune de ces pistes pourrait donner une autre ampleur au projet et lui permettre de passer de l'état de prototype à l'état de logiciel commercialisable sur le marché.

Nous avons différencié deux grands ensembles de pistes de recherche. Le premier comprend les moyens permettant d'alléger le travail de l'utilisateur dans la pondération, à chaque phrase, des différents descripteurs du système. Le second correspond aux autres aspects du logiciel.

7.2.1. Faciliter le travail préliminaire de l'utilisateur

Les pistes de recherche proposées ici concernent principalement la manière dont le système pourrait simplifier le travail préliminaire (création des fichiers 'Database.in' et 'Diary.in') que doit effectuer l'utilisateur avant de pouvoir utiliser le système.

En effet, un premier travail serait de faciliter l'assignation des valeurs des descripteurs pour chaque phrase de la base de données. Comme on l'a vu plus haut, certains utilisateurs ont des difficultés à évaluer le degré de pertinence d'une phrase par rapport à son descripteur. Il serait donc intéressant de trouver une autre manière de faire.

- **Une première façon** de faire serait de remplacer l'assignation de valeurs numériques par l'assignation de labels linguistiques. Ainsi, au lieu d'utiliser des nombres entre 1 et 20, l'utilisateur marquerait le degré de pertinence d'une phrase par rapport à un descripteur par des expressions telles que *fort lié*, *très fort lié*, *extrêmement fort lié*, etc. Le changement serait minime dans la conception du logiciel. En effet, il suffirait de faire correspondre ces labels linguistiques aux chiffres (entre 1 et 20) pour que le système ne subisse aucun changement. La Figure 7.5 nous en donne une première idée.

  Ce type d'assignation linguistique permettrait à l'utilisateur de mieux situer la phrase dans son contexte qu'en utilisant une assignation numérique.

1	Pas de lien du tout	11	Lien un peu plus que moyen
2	Extrêmement très très très peu de lien	12	Fort lié
3	Extrêmement très très peu de lien	13	Très fort lié
4	Extrêmement très peu de lien	14	Très très fort lié
5	Extrêmement peu de lien	15	Très très très fort lié
6	Très très très peu de lien	16	Extrêmement fort lié
7	Très très peu de lien	17	Extrêmement très fort lié
8	Très peu de lien	18	Extrêmement très très fort lié
9	Peu de lien	19	Extrêmement très très très fort lié
10	Lien moyen	20	Complétement et intégralement lié

Figure 7.5 : Correspondance des nombres avec des labels linguistiques

- En allant dans la même direction, **une autre façon** de procéder serait d'utiliser un objet interactif de contrôle : l'échelle. L'échelle est un objet de contrôle permettant à l'utilisateur d'introduire une valeur numérique, par ajustement d'un indicateur, à une position spécifique, sur une ligne graduée. La figure 7.4 nous montre ce que pourrait être un pareil objet.

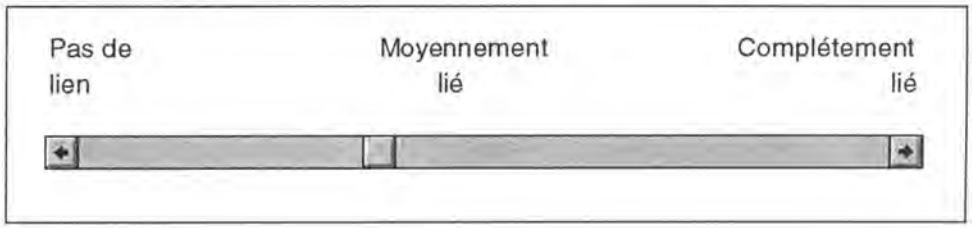

Figure 7.6 : Utilisation d'une *échelle*

L'utilisateur assignerait le degré de pertinence d'une phrase par rapport à son descripteur, en bougeant le curseur de l'échelle à la position qu'il considère comme juste. Cela permettrait à l'utilisateur de mieux visualiser le degré de pertinence qu'il assigne à une phrase.

- **Une toute autre manière** de faire serait de reporter le problème sur le système lui-même plutôt que de le laisser sur les épaules de l'utilisateur. Celui-ci pourrait simplement encoder ses phrases dans le fichier 'Database.in', sans qu'aucune association avec des descripteurs ne soit faite. Ainsi, ce serait au système de concevoir une étape préliminaire à son utilisation. Il devrait analyser les phrases de la base de données et les associer lui-même directement aux descripteurs du système.

Bien sûr, beaucoup de recherches et de tests devront être effectués dans ce domaine-là, puisque la compréhension du langage par l'ordinateur n'est pas en mesure d'être faite. Les tests devront aller dans le sens de *simuler la compréhension du langage* par des techniques diverses.

On pourrait, par exemple, assigner aux descripteurs de sujets une série de mots leur appartenant. Ainsi, le descripteur sujet *sport* se verrait assigner les mots : piscine, course, effort, football, gymnastique, muscle, crampe, soif, condition physique, etc. Et lorsque le système rencontre un de ces mots dans une phrase de la base de données, il lui assignerait une valeur élevée pour le descripteur sport. Bien sûr, de nombreuses recherches devront être faites dans ce sens pour valider cette technique. De plus, pour les autres descripteurs, cette technique serait plus difficilement adaptable.

7.2.2. D'autres pistes

Voici encore quelques pistes pour les travaux futurs à propos de ce projet. Ces pistes concernent tant des ajouts que l'on pourrait apporter au système que des modifications qui le rendraient peut-être plus efficace. En tout état de cause, chacun des points cités ci-dessous doit faire l'objet de recherches approfondies et de nombreux tests, avant qu'il ne soit validé.

Actuellement, quatre phrases sont proposées par le système à chaque prédiction. Si l'utilisateur veut davantage de phrases ou en veut d'autres, il doit demander explicitement une nouvelle recherche dans la base de données (par l'option « More »). On pourrait imaginer une modification du logiciel, en proposant à l'utilisateur et à chaque prédiction, l'ensemble des phrases qui serait au-dessus d'un certain niveau de correspondance avec la phrase courante. Ainsi, l'utilisateur aurait normalement en une fois toutes les phrases associées à la phrase principale. L'utilisation d'une liste déroulante serait ici de rigueur.

Il serait aussi probablement possible d'inclure dans le système l'outil suivant : en fin de conversation, le système pourrait évaluer la pertinence de chaque descripteur dans la prédiction des phrases. En effet, il se peut que certains descripteurs ne jouent quasi jamais de rôle prédominant dans le fait de choisir une phrase plutôt qu'une autre. Ces descripteurs n'ont donc aucune raison de rester dans le système.

D'autres études et investigations pourraient être également faites dans les domaines suivants :

- Déterminer le nombre et le type de descripteurs optima pour que l'utilisation du système soit aussi efficace que possible.
- Déterminer le poids optimum à allouer aux descripteurs lorsqu'ils se trouvent dans le fichier 'Diary.in'.
- Des tests devront être expérimentés avec d'autres ensembles de descripteurs.

Enfin, il est excessivement important d'avoir une interface intuitive et facile à utiliser pour ce genre de logiciel. Les travaux futurs devront donc aller dans ce sens. Ainsi d'autres tests et développements devront être réalisés sur l'interface de ce système pour essayer de l'améliorer.

Conclusion

Le but de ce projet était d'explorer la possibilité d'appliquer la théorie des ensembles flous dans la construction d'un système d'aide à la communication pour les personnes souffrant de problèmes de langage. Ce projet a connu deux grandes étapes : il a consisté d'abord en un ensemble de recherches dans le domaine de la théorie des ensembles flous et dans son application dans les prothèses de communication. La seconde étape du projet eut pour objectifs la conception et la réalisation d'un système complet d'aide à la communication.

Dans la première partie de ce travail, nous avons ainsi résumé les principales recherches connues et répertoriées dans le secteur étudié. C'est ainsi que le premier chapitre nous a révélé à quel point la communication était importante dans la vie de tous les jours. Nous avons également décrit les caractéristiques et les qualités que devrait avoir tout système d'aide à la communication. Nous nous sommes ensuite attardés aux personnes souffrant de problèmes de langage et aux solutions à envisager pour les aider. Dans le deuxième chapitre, nous avons décrit les fondements de la théorie des ensembles flous qui seront exploités plus tard dans la conception du système d'aide à la communication. Nous avons terminé cette première partie en faisant un rapide inventaire des différents outils et systèmes déjà réalisés dans le domaine qui nous occupe.

Dans la seconde partie, nous nous sommes attachés à présenter le système Fuzzy-Chat. C'est un système d'aide à la communication pour personne souffrant de problèmes de langage. Il comporte notamment un système de prédiction où la recherche d'information se base sur des éléments de la théorie des ensembles flous. C'est ainsi que, dans le quatrième chapitre, nous avons décrit l'ensemble des fonctionnalités du système

conçu. De la même manière, nous avons justifié la construction de son interface dans le cinquième chapitre. Après un chapitre plus technique, nous avons achevé ce travail, dans le septième chapitre, par l'évaluation du système construit et par un bref inventaire de différentes pistes de recherches pour des travaux futurs dans ce domaine.

La phase de documentation du projet a permis de dégager les différents objectifs que tout système d'aide à la communication devrait poursuivre, à savoir : *expression des besoins, transfert d'information, relations sociales et étiquettes sociales.* Elle a montré également que beaucoup de systèmes ne répondent souvent qu'au premier but (*expression des besoins*). De même, l'étude des systèmes d'aide à la communication existants a mis le doigt sur deux grosses faiblesses : 1) le rythme des conversations est trop lent; 2) les systèmes mettent, à charge de l'utilisateur, une partie du travail de la recherche d'information (par exemple, au début de chaque conversation, il doit rentrer des éléments permettant au système de proposer les objets de conversations adéquats). Finalement, les différentes évaluations de ces projets (avec des personnes handicapées) ont démontré l'importance de l'interface et de la facilité d'utilisation que devrait avoir un système d'aide à la communication.

C'est principalement sur ces bases que le système d'aide à la communication, FuzzyChat, a été construit. Sa première grande étape a été la réalisation du système de prédiction (*system retrieval*). Il a utilisé des éléments de la théorie des ensembles flous pour lier les phrases de la base de données entre elles. Une lourde tâche est demandée à l'utilisateur dans la phase préliminaire de l'utilisation du système, dans la mesure où il doit encoder l'ensemble de ses phrases dans la base de données. Il le fait en associant chaque phrase à une série de descripteurs. Cela réalisé, le système se chargera de soumettre un ensemble de phrases à l'utilisateur. Celui-ci sélectionnera donc la phrase voulue, ce qui produira une sortie parlée à l'aide d'un synthétiseur de voix. Ces phrases sont proposées de telle manière qu'elles fournissent un prolongement cohérent à la communication. Les prédictions tiennent donc compte non seulement des liens entre les descripteurs de chaque phrase mais également de deux éléments supplémentaires : 1) des fréquences d'utilisation des phrases dans les précédentes conversations; 2) des informations contenues dans *l'agenda* de l'utilisateur, ce qui permet d'accorder plus d'importance à certains descripteurs lors des prédictions. Le logiciel inclut également des options de relance et de désactivation du système de prédiction, de backtracking et d'aide.

La seconde étape de la conception du système FuzzyChat comporte la création de l'interface du système. Il a été construit en tenant compte le plus possible des facteurs humains qui jouent un rôle fondamental dans la construction d'un système informatique

interactif. De plus, il a inclu deux écrans d'aide pour faciliter les premiers maniements du récent utilisateur, ainsi qu'une aide en ligne.

La dernière étape de la construction du logiciel fut la suivante : deux outils supplémentaires ont été inclus au système pour le rendre le plus complet possible et pour combler les lacunes qu'aurait eues le système de prédiction seul. Le premier outil donne à l'utilisateur la possibilité de créer manuellement du texte à l'aide du clavier. Le second outil permet à l'utilisateur de *dire* (via le synthétiseur de voix) des petits commentaires rapides et répétitifs.

Une première évaluation du système a été réalisée et a donné des premiers résultats positifs. Cependant d'autres évaluations, et notamment avec des personnes handicapées, sont nécessaires pour confirmer ses premiers résultats. Elles vont avoir lieu dans le courant de l'année prochaine au MicroCentre de l'université de Dundee.

Ces premiers résultats ont confirmé le fait qu'un système d'aide à la communication, utilisant la théorie des ensembles flous dans sa recherche d'information, peut être utile pour les problèmes de communication. Le stockage au préalable des données conversationnelles, dans un système de prédiction, est fort efficace dans la construction rapide de conversations.

En ce qui concerne les buts que doivent poursuivre les systèmes d'aide à la communication, deux ont été atteints avec succès : *transfert d'information* et *relations sociales*. Malheureusement, le système est moins bien adapté dans le cas de *l'étiquette sociale*. Il en est de même pour le but *expression des besoins*, but non prioritaire du projet puisqu'il est déjà bien pris en compte par les systèmes actuels d'aide à la communication.

Un des points forts du logiciel réside dans l'utilisation du système de prédiction, permettant d'alléger fortement la charge cognitive de l'utilisateur et favorisant la participation plus naturelle de ce dernier à la conversation. La rapidité des conversations ainsi que l'interface représentent aussi deux points positifs du programme, relevés lors de l'évaluation.

Un premier élément négatif de ce système est bien sûr la phase d'encodage préliminaire que doit effectuer tout nouvel utilisateur. Ceci est dû au fait que le système de prédiction a besoin de données personnelles pour s'exécuter. Cette phase d'encodage est lourde et peut être un obstacle radical à l'utilisation du système. L'éventuelle possibilité de créer manuellement du texte au clavier constitue un autre point négatif. Cela ralentit

considérablement le rythme de la conversation. Heureusement des systèmes existent en vue d'augmenter la vitesse de frappe d'un utilisateur peu expérimenté.

Les travaux futurs en lien avec ce projet devront prévoir d'abord de nouvelles évaluations approfondies du système. Celles-ci permettront de confirmer, d'infirmer ou de nuancer les conclusions tirées jusqu'à présent. La seconde étape essentielle du suivi sera de trouver une technique facilitant le problème d'encodage préliminaire des objets conversationnels. La réussite de cette étape est cruciale pour faciliter l'utilisation de ce système d'aide et ainsi pour améliorer la qualité et l'efficacité de ce logiciel. Il est donc essentiel que la recherche dans ce domaine soit poursuivie par de nouvelles équipes d'informaticiens.

Bibliographie

[Alm, 92] Norman Alm, John L. Arnott, Alan F. Newell (November 1992), « An integrated, multi-level communication system for physically impaired non-speaking children and adults », *First International Conference of the Saudi Association for Handicapped Children* (Riyadh, Saudi Arabia).

[Alm, 93a] Norman Alm, John Todman, Leona Elder, A. F. Newell (April 1993), « Computer Aided Conversation for Severely Physically Impaired Non-speaking People », *INTERCHI'93 Conference proceedings, Conference on Human Factors in Computing Systems* (Amsterdam, The Netherlands), p 236-241.

[Alm, 93b] Norman Alm (1993), « The Development of Augmentative and Alternative Communication Systems to Assist with Social Communication », *Technology and disability*, 2 (3), p. 1- 18.

[Beukelman, 92] David R. Beukelman, Pat Mirenda (1992), *Augmentative and Alternative Communication : Management of severe communication disorders in children and adults*, Paul H. Brookes Publishing Co.

[Bodart, 90] François Bodart (1990), *Cours introductif aux interfaces homme-machine* », Institut d'informatique, Université de Namur (FUNDP).

[Broumley, 90] Liz Broumley, John L. Arnott, Alistair Y. Cairns, Alan F. Newell (August 1990), « TalksBack : An application of AI techniques to a

communication prosthesis for the non-speaking », *9th European Conference on Artificial Intelligence* (Stockholm, Sweden), p 117-119.

[Cox, 94] Earl Cox (1994), *The fuzzy systems handbook : a practitioner's guide to building, using and maintaining fuzzy systems*, Academic Press Limited.

[De Mori, 83] Renato De Mori (1983), *Computer models of speech using fuzzy algorithms*, Plenum Press (New York).

[Gardeazabal, 95] Luis Gardeazabal, Julio Gonzalez-Abascal, Nestor Garay, Alberto Postigo, Santiago Urigoitia, Maria Jesus Martinez de Morentin (April 1995), « Design and integration of alternative and augmentative interfaces under Windows environment », *The European context for assitive technology, Proceedings of the 2nd TIDE Congress*, Vol. 1, p 340-343.

[Gupta, 79] Madan M. Gupta, Rammohan K. Ragade, Ronald R. Yager (1979), *Advances in fuzzy set theory and applications*, North-Holland Publishing Compagny.

[Jacquard, 94a] Propos d'Albert Jacquard recueillis par Jacqueline Huber (Juillet-août 1994), « Plus est en moi », *Messages du secours catholique*, N° 472.

[Jacquard, 94b] Propos d'Albert Jacquard recueillis par Myriam Katz (21 décembre 1994), « Albert Jacquard ou l'Utopie », *Le Ligueur*, N°50, p 8.

[Marlborough, 90] Mary Marlborough Lodge (1990), *Equipment for disabled people*, G.M. Cochrane MA, FRCP, Seventh edition.

[Morris, 92] Corinne Morris, Alan Newell, Lynda Booth, Ian Ricketts, John Arnott (1992), « Syntax PAL : A system to improve the written syntax of language-impaired users », *Assistive technology*, Vol. 4, No 2, p51-59.

[Newell, 91] Alan F. Newell, John L. Arnott, Lynda Booth, William Beattie, Bernadette Brophy, Ian W. Ricketts (1991-1992), *The use of the 'PAL' word prediction system on the quality and quantity of text generation*, The MicroCentre, University of Dundee (Scotland).

[Newell, 92] Alm N. Newell A. F., Arnott J. L., « Prediction and conversational momentum in an augmentative communication system », *Communication of the ACM*, 35(5), p 46-57.

[Nicol, 93] Mark Nicol, Norman Alm, John Arnott (June 1993), « The application of fuzzy set theory to the storage and retrieval of conversational texts in an augmentative communication system », *RESNA '93 Annual Conference*.

[Ricketts, 91] Ian W Ricketts, Lynda Booth (December 1991), *PAL — A Predictive Word Processing System*, MicroCentre, University of Dundee (Scotland).

[Shneiderman, 86] Ben Shneiderman (1986), *Designing the User Interface : Strategies for Effective Human-Computer Interaction*, Addison-Wesley publishing compagny.

[Smith, 84] Smith, Sidney L., Mosier, Jane N. (September 1984), *Design guidelines for user-system interface software*, Report ESD-TR-84-190, The Mitre Corp., Bedford, MA.

[Vanderdonckt, 92] Jean Vanderdonckt (octobre 1992), *Corpus ergonomique minimal des applications de gestion*, Institut d'informatique, Université de Namur (FUNDP).

[Waller, 91] Annalu Waller, Liz Broumley, Alan F. Newell, Norman Alm (1991), « Predictive retrieval of conversational narratives in an augmentative communication system », *RESNA 14th Annual conference* (Kansas City).

[Waller, 93] Waller A., Dennis F. (January 1993 - December 1993), « The development of 'TalksBac' : A Computer-Based Communication System », *Report to the Leverhume Trust.*, MicroCentre, University of Dundee, Scotland.

[Zadeh, 65] L. A. Zadeh (1965), « Fuzzy sets », *Inform. Contr.*, vol. 8, p 338-353.

[Zadeh, 70] Lotfi A. Zadeh (January 1970), « Outline of a New Approach to the Analysis of Complex Systems and Decision Processes », *IEEE Transactions on systems, man, and cybernetics*, Vol. SMC-3, No 1, p 28-44.

Annexe A

Code source du système FuzzyChat

A.1. Main file : Coordina.cpp

```
////////////////////////////////////////////////////////////////////////
// -- Coordinator file (main)                                         //
////////////////////////////////////////////////////////////////////////


// Include files (standard)

#include <iostream.h>      // Declares the basic C++ streams(I/O) routines.
#include <constream.h>     // Declares the C++ classes and methods to support   console output.
#include <fstream.h>       // Declares the C++ stream classes that support file input and output.
#include <stdlib.h>        // Declares several commonly used routines.
#include <time.h>          // Defines a structure filled in by the time-conversion routines.
#include <string.h>        // Declares several string-manipulation and memory manipulation routines.
#include <graphics.h>      // Declares prototypes for the graphics functions.
#include <ctype.h>         // For the function : 'isspace()'
#include <dos.h>           // Defines various constants and gives declarations needed for DOS and 8086-
                           // specific calls.

// Global Constants

#define Max_conversation_item      140      // Maximum length for a conversation item.
#define Max_subject                10       // Maximum number of subjects descriptors.
#define Max_purpose                 8       // Maximum of purpose descriptors.
#define Max_person                  8       // Maximum of person descriptors.
#define Max_item                  100       // Maximum of conversation item.
#define Fzy_recently_used          12       // Weighting for the item last used.
```

```
#define Fzy_augmentation              1      // Value added to Used after each search (until variable
                                             // reaches 12).
#define Max_length_descriptors       15      // Length maximum for a descriptor
#define  Weight_diary              0.30      // Weight to add to a descriptor. (from the 'diary.in' file)

// include files (objects)

#include "backtrac.cpp"    // Declares the C++ classes and methods for the 'backtrack_list' object.
#include "conversa.cpp"    // Declares the C++ classes and methods for the 'conversation_item_record'
                           // object
#include "descript.cpp"    // Declares the C++ classes and methods for the 'descriptors' object.
#include "fzy_unit.cpp"    // Declares the C++ classes and methods for the 'FDB' object
#include "graphobj.cpp"    // Declares the C++ classes and methods for the 'Rectangle' object and
                           // 'Button' object.
#include "msmouse.hpp"     // Header file for the mouse object.
#include "msmouse.cpp"     // Declares the C++ classes and methods for the 'MouseObject' object.
#include "mcursor.cpp"     // Declares shapes for the mouse pointers.

// Global Variables

Conversation_Item_Record   Database[Max_item];                    // object
                // Database of the software.
Descriptors                Subject_descriptor[Max_subject];       // object
                // Weight of each subject descriptors.
Descriptors                Purpose_descriptor[Max_purpose];      // object
                // Weight of each purpose descriptors.
Descriptors                Person_descriptor[Max_person];         // object
                // Weight of each person descriptors.
Backtrack_List             Backtrack;                            // object
                // Object for the backtracking.
int                        Nb_item;
                // Number of conversation item in the database.
int                        Nb_subject;
                // Number of subject descriptors.
int                        Nb_purpose;
                // Number of purpose descriptors.
int                        Nb_person;
                // Number of person descriptors.
int                        Freeze;
                // if = 0, no freeze,if =, freeze.
int                        BoxPrinc;
                // Index in the array 'Database' of the Principal Box.
int                        Box1;
                // Index in the array 'Database' of the Box 1.
int                        Box2;
                // Index in the array 'Database' of the Box 2.
int                        Box3;
                // Index in the array 'Database' of the Box 3.
int                        Box4;
                // Index in the array 'Database' of the Box 4.
int                        Back_down;
                // If Back-Down is possible, = 1 otherwise = 0.
int                        Back_up;
                // If Back-Up is possible, =1 otherwise =0.
short                      COM=1;
                // Declaration of the port for the speech synthesiser.
```

```
short                       Synthesiser=0;
                // Synthesiser is On if =1, is Off if =0.

// Global Variables (graphics)

Rectangle BoxPrincG (14,3,1,20,20,490,90,0);
                // Variable for the principal box (box=0).
Rectangle Box1G (14,1,2,20,110,430,145,1);
                // Variable for the box 1.
Rectangle Box2G (14,1,2,40,170,450,205,1)
                // Variable for the box 2
Rectangle Box3G (14,1,2,60,230,470,265,1)
                // Variable for the box 3
Rectangle Box4G (14,1,2,80,290,490,325,1);
                // Variable for the box 4.
Button More  ("More",10,3,7,80,375,45,20);
                // Variable for the 'More' button.
Button FreezeG("Freeze",10,3,7,190,375,45,20);
                // Variable for the 'Freeze' button.
Button Unfreeze("UnFreeze",10,3,7,190,375,45,20);
                // Variable for the 'Unfreeze' button.
Button BD ("",10,3,7,300,375,45,20);
                // Variable for the '<' button.
Button BU ("",10,3,7,410,375,45,20);
                // Variable for the '>' button.
Button Help ("Help",10,3,7,573,375,45,20);
                // Variable for the 'Help' button.
Rectangle ChBox(10,3,1,530,20,620,290,0);
                // Variable for the 'Chat Box'.
Rectangle Exit(10,3,2,530,462,620,476,0);
                // Variable for the 'Exit' button.
Rectangle Text(14,1,2,20,420,620,450,0);
                // Variable for the Unique text.

MouseObject Mouse;
                // Mouse object.

// include files (modules)

#include "synth.cpp"        // Functions of the 'Synthesiser' module.
#include "graphic.cpp"      // Functions of the 'Graphics' module.
#include "finish.cpp"       // Functions of the 'Finishing' module.
#include "textchat.cpp"     // Functions of the 'Unique text' and 'ChatBox' module.
#include "unloadin.cpp"     // Functions of the 'Unloading' module.
#include "speechbox.cpp"    // Functions of the 'Speechbox' module.
#include "help.cpp"         // Functions of the 'help' module.


// Main function.

void main ()
{
        clrscr();
                // clear the screen.

        Database_file();
```

```
        // Unload the 'Database.in' file and fill 'Database','Subject_descriptor',
        // 'Purpose_descriptor' and 'Person_descritor' variables.

Diary_file();
        // Unload the 'Diary.in' file and fill 'Database','Subject_descriptor',
        // 'Purpose_descriptor' and 'Person_descritor' variables.
SetUp();
        // Unload the 'Setup.in' file and update 'COM' and 'Synthesiser' variables.
InitGraph();
        // Initialise the graphic mode.
Boites_normales();
More.Show();
FreezeG.Show();
BackDownNet();
BackUpNet();
Help.Show();
Text.Show();
Exit.Show();
Exit.WriteCenter("E  x  i  t");
SetHelpLine();
Chat_draw();
        // Creation of the interface.
Mouse.Setup (Graphics);
if (!Mouse.Operating())
{
        closegraph();
        cout << "\nError : Mouse not detected";
        exit(1);
}
        // Initialise the Mouse.

Mouse.TurnOn();
        // Activation of the mouse.

Initialisation();
        // Initialisation of some global variables and first search in the database.

Conversation_file("","");
        // Beginning of the filling of the 'Conversa.out' file.

unsigned E;
int Mx, My;
int Cursor=1;  // cursor pointers : 1=Cross, 2=Mouth, 3=Hand, 4=IBeam.
int end=0;
        // Local variables for the management of the mouse.

do
        // main loop
{
        E = Mouse.Event(Mx,My);
                // Waiting for a mouse event - (Mx,My) = position of the mouse

        int box = Which_box(Mx, My, Cursor, Back_down, Back_up);
                // Initialise 'box'according to the mouse event.

        if (E==LMouseDown)
```

```
        // if left button (of the mouse) pushed
        switch (box)
        {

// if the user clicks on the Principal Box (Box 0).
        case 0 : Mouse.SetGCursor(OMouthCursor);
                        Mouse.WaitForEvent(LMouseUp,Mx,My);
                        Mouse.SetGCursor(MouthCursor);
                        Find_Clotest(0);
                        break;

// if the user clicks on the Box 1.
        case 1 : Mouse.SetGCursor(OMouthCursor);
                        Mouse.WaitForEvent(LMouseUp,Mx,My);
                        Mouse.SetGCursor(MouthCursor);
                        Find_Clotest(1);
                        break;

// if the user clicks on the Box 2.
        case 2 : Mouse.SetGCursor(OMouthCursor);
                        Mouse.WaitForEvent(LMouseUp,Mx,My);
                        Mouse.SetGCursor(MouthCursor);
                        Find_Clotest(2);
                        break;

// if the user clicks on the Box 3.
        case 3 : Mouse.SetGCursor(OMouthCursor);
                        Mouse.WaitForEvent(LMouseUp,Mx,My);
                        Mouse.SetGCursor(MouthCursor);
                        Find_Clotest(3);
                        break;

// if the user clicks on the Box 4.
        case 4 : Mouse.SetGCursor(OMouthCursor);
                        Mouse.WaitForEvent(LMouseUp,Mx,My);
                        Mouse.SetGCursor(MouthCursor);
                        Find_Clotest(4);
                        break;

// if the user clicks on the 'More' button.
        case 5 : Mouse.Hide();
                        More.ChangeColor(2);
                        More.Show();
                        Mouse.Show();
                        Mouse.WaitForEvent(LMouseUp,Mx, My);
                        Mouse.Hide();
                        More.ChangeColor(10);
                        More.Show();
                        Mouse.Show();
                        More_button();
                        break;

// if the user clicks on the 'Freeze/Unfreeze' button.
        case 6 : if (Freeze==0)                    // if no freezing.
                        {
                                Mouse.Hide();
```

```
                    FreezeG.ChangeColor(2);
                    FreezeG.Show();
                    Mouse.Show();
                    Mouse.WaitForEvent(LMouseUp,Mx,My);
                }
                else
                {                                   // if freezing.
                    Mouse.Hide();
                    Unfreeze.ChangeColor(2);
                    Unfreeze.Show();
                    Mouse.Show();
                    Mouse.WaitForEvent(LMouseUp,Mx,My);
                }
                Freezer();
                break;

// if the user clicks on the '<' button.
        case 7 : if (Back_down==1)          // if the button is active.
                {
                    Mouse.Hide();
                    BackDownFuzzy();
                    Mouse.Show();
                    Mouse.WaitForEvent(LMouseUp,Mx,My);
                    Mouse.Hide();
                    BackDownNet();
                    Backtrack_down();
                    Mouse.Show();
                }
                break;

// if the user clicks on the '>' button.
        case 8 : if (Back_up==1)            // if the button is active.
                {
                    Mouse.Hide();
                    BackUpFuzzy();
                    Mouse.Show();
                    Mouse.WaitForEvent(LMouseUp,Mx,My);
                    Mouse.Hide();
                    BackUpNet();
                    Backtrack_up();
                    Mouse.Show();
                }
                break;

// if the user clicks on the 'help' button.
        case 9 : Mouse.Hide();
                Help.ChangeColor(2);
                Help.Show();
                Mouse.Show();
                Mouse.WaitForEvent(LMouseUp,Mx,My);
                Mouse.Hide();
                Help.ChangeColor(10);
                Help.Show();
                Mouse.Show();
                help(Freeze, Back_down, Back_up);
                break;
```

```
                // if the user clicks on the box : text.
                        case 10 :Mouse.Hide();
                                        TextBox();
                                        Mouse.Show();
                                        break;

                // if the user clicks on the ChatBox
                        case 11 :Mouse.SetGCursor(GMouthCursor);
                                        Mouse.WaitForEvent(LMouseUp,Mx,My);
                                        Mouse.SetGCursor(MouthCursor);
                                        Chat_speak(Mx,My);
                                        break;

                // if the user clicks on the 'exit' button.
                        case 12 :end=1;
                                        break;

                        }
                }
        while (!end);
                // While the user has not pressed on the exit button.

        closegraph();
                // Shuts down the graphics system.

        Conversation_file("EXIT BUTTON","");
                // Finishing of the filling of the 'Conversa.out' file.

        Ajustement_freq();
                // Update the frequency in the 'Database.in' file.
}
```

A.2. Object files

A.2.1. Backtrack.cpp

```
////////////////////////////////////////////////////////////////////////////////
// -- 'Backtrack' Object.                                                     //
// -- Management of the backtracking in the software.                         //
////////////////////////////////////////////////////////////////////////////////

#ifndef _BACKTRAC.CPP
#define _BACKTRAC.CPP

#include <stdlib.h>

        // one element of the backtrack chain.
struct Element_Backtrack
```

```
{
        Element_Backtrack * Next;  // double chain list.
        Element_Backtrack * Prec;  //
        int BPrinc;                // index of the Principal Box in the array 'Database'.
        int B1;                    // index of the Box 1 in the array 'Database'.
        int B2;                    // index of the Box 2 in the array 'Database'.
        int B3;                    // index of the Box 3 in the array 'Database'.
        int B4;                    // index of the Box 4 in the array 'Database'.
};


        // backtrack chain
class Backtrack_List
{
        Element_Backtrack * courant;        // pointer on the courant element.
public :
        Backtrack_List ()                   // constructor
                { courant=NULL; }
        void Add (int, int, int, int, int);
                                            // Add of one element to the chain.
        void GivePrec(int &, int &, int&, int&, int&);
                                            // Back-down.
        void GiveNext(int &, int &, int&, int&, int&);
                                            // Back-up.
        int First();
                                            // 1 if courant is the first on the list, 0 else.
        int Last();
                                            // 1 if courant is the last on the list, 0 else.
        ~Backtrack_List ();
                                            // Destructor of the list.
};

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

void Backtrack_List::Add (int P, int b1, int b2, int b3, int b4)
// Add one element to the backtrack chain.
{
        Element_Backtrack * pt1;
        Element_Backtrack * pt2;
        Element_Backtrack * cel = new Element_Backtrack;
        cel -> Next = NULL;
        cel -> BPrinc=P;
        cel -> B1=b1;
        cel -> B2=b2;
        cel -> B3=b3;
        cel -> B4=b4;

                // if no element in the backtrack chain.
        if (courant==NULL)
        {
                courant = cel;
                cel->Prec=NULL;
        }
```

```
                // if one or more elements in the backtrack chain.
        else
        {
                pt1 = courant -> Next;
                while (pt1!=NULL)
                {                                       // Delete every element in the backtrack chain
                        pt2=pt1;                        // which are after the 'courant' pointer.
                        pt1=pt1->Next;                  //
                        delete (pt2);
                }
                courant->Next=cel;
                cel->Prec = courant;
                courant = cel;
        }
}

////////////////////////////////////////////////////////////////////////////////

void Backtrack_List::GivePrec (int &P, int &b1, int &b2, int &b3, int &b4)
// Return the index of the previous element in the backtrack chain for each speech box
// Move the 'courant' pointer (one element before).
{
        courant=courant->Prec;
        P=courant->BPrinc;
        b1=courant->B1;
        b2=courant->B2;
        b3=courant->B3;
        b4=courant->B4;
}

////////////////////////////////////////////////////////////////////////////////

void Backtrack_List::GiveNext (int &P, int &b1, int &b2, int &b3, int &b4)
// Return the index of the next element in the backtrack chain for each speech box
// Move the 'courant' pointer (one element after).
{
        courant=courant->Next;
        P=courant->BPrinc;
        b1=courant->B1;
        b2=courant->B2;
        b3=courant->B3;
        b4=courant->B4;
}

////////////////////////////////////////////////////////////////////////////////

int Backtrack_List::First()
```

```
// Return 0 if there is a element before the 'courant' pointer
// Return 1 otherwise.
{
        if (courant->Prec==NULL) return 1;
        else return 0;
}


/////////////////////////////////////////////////////////////////////////////////////


int Backtrack_List::Last()
// Return 0 if there is an element after the 'courant' pointer
// Return 1 otherwise.
{
        if (courant->Next==NULL) return 1;
        else return 0;
}


/////////////////////////////////////////////////////////////////////////////////////


Backtrack_List::~Backtrack_List()
// Destruction of the backtrack chain
{
        Element_Backtrack * pt;
        pt=NULL;

                // go to the last element of the backtrack chain.
        while (courant!=NULL)
        {
                pt = courant;
                courant=courant->Next;
        }
        courant=pt;

                // delete all elements of the backtrack chain.
        while (courant!=NULL)
        {
                pt=courant;
                courant=courant->Prec;
                delete (pt);
        }
}


#endif   // _BACKTRACK.CPP
```

A.2.2. Conversa.cpp

```
////////////////////////////////////////////////////////////////////////////////
// -- 'Conversation_Item_Record' Object.                                      //
// -- Database of the software with the 'Descriptor' Object.                  //
////////////////////////////////////////////////////////////////////////////////
#ifndef _CONVERSA.CPP
#define _CONVERSA.CPP

#include <string.h>


class Conversation_Item_Record
{
        char *   Conversation_Item;
                                        // Conversation Item
        short    Subject[Max_subject+1];
                                        // Value of the subject descriptors
        short    Purpose[Max_purpose+1];
                                        // Value of the purpose descriptors
        short    Person[Max_person+1];
                                        // Value of the person descriptors
        short    Frequency;
                                        // Value of the frequency
        short    Used;
                                        // Value of the recent Used
        short    Utilisation;
                                // if 0, no selected, no used
                                // if -1, no selected, used    (for the update of the frequency)
                                // if 1, selected, used.
public :
        Conversation_Item_Record ();
                // Constructor : Initialisation of 'Used' and 'Utilisation'
        void InitSubject (short weight, int pos);
                // Initialisation of 'Subject[pos]' with weight
        void InitPurpose (short weight, int pos);
                // Initialisation of 'Purpose[pos]' with weight
        void InitPerson (short weight, int pos);
                // Initialisation of 'Person[pos]' with weight
        void InitFrequency (short freq);
                // Initialisation of 'Frequency'
        void InitConversationItem (char * ph);
                // Initialisation of 'Conversation_Item'
        char* GiveConversation();
                // Return 'Conversation_Item'
        void ChangeUsed (int u);
                // Change 'Used' : Used = u
        short GiveUsed();
                // Return 'Used'
        void ModifyUsed (short u);
                // Modify 'Used' : Used = Used + u
        int GiveUtilisation ();
                // Return 'Utilisation'
        void ChangeUtilisation (short change);
                // Change 'Utilisation'
```

```
        int GiveSubject (int nb);
                // Return 'Subject[nb]'
        int GivePurpose (int nb);
                // Return 'Purpose[nb]'
        int GivePerson (int nb);
                // Return 'Person[nb]'
        int GiveFrequency ();
                // Return 'Frequency'
};

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

Conversation_Item_Record::Conversation_Item_Record()
{
        Utilisation=0;
        Used=Fzy_recently_used;
}

void Conversation_Item_Record::InitSubject (short weight, int pos)
{
        Subject[pos] = weight;
}

void Conversation_Item_Record::InitPurpose (short weight, int pos)
{
        Purpose[pos] = weight;
}



void Conversation_Item_Record::InitPerson (short weight, int pos)
{
        Person[pos] = weight;
}

void Conversation_Item_Record::InitFrequency (short freq)
{
        Frequency = freq;
}

void Conversation_Item_Record::InitConversationItem (char * ph)
{
        Conversation_Item = new char[strlen(ph)+1];
        strcpy(Conversation_Item,ph);
}

char * Conversation_Item_Record::GiveConversation()
{
        return (Conversation_Item);
}

void Conversation_Item_Record::ChangeUsed (int u)
{
        Used = u;
}
```

```
short Conversation_Item_Record::GiveUsed()
{
        return (Used);
}

void Conversation_Item_Record::ModifyUsed (short u)
{
        Used=Used + 1;
}

int Conversation_Item_Record::GiveUtilisation ()
{
        return (Utilisation);
}

void Conversation_Item_Record::ChangeUtilisation (short change)
{
        Utilisation = change;
}

int Conversation_Item_Record::GiveSubject (int nb)
{
        return (Subject[nb]);
}

int Conversation_Item_Record::GivePurpose (int nb)
{
        return (Purpose[nb]);
}

int Conversation_Item_Record::GivePerson (int nb)
{
        return (Person[nb]);
}

int Conversation_Item_Record::GiveFrequency ()
{
        return (Frequency);
}

#endif   // _CONVERSA.CPP
```

A.2.3. Data_fzy.hpp

```
////////////////////////////////////////////////////////////////////////////
// -- Data for the fuzzy algorithm and the fuzzy object (FDB).            //
////////////////////////////////////////////////////////////////////////////
#ifndef _DATA_FZY.HPP
#define _DATA_FZY.HPP

        // Definition of boolean value
#define TRUE        1
#define FALSE       0
```

```
        // Number of vector in the 'FDB' object (FDBvector)
#define VECMAX              257

        // Default domain
#define domain1 -2
#define domain2 22

        // Largeur of a normal(frequency=3) triangular set.
#define largeur 10

        // Macro : definition of minimum
#define min(a,b)    (((a) < (b)) ? (a) : (b))

        // definition of a boolean type
typedef int                         bool;


#endif   // _DATA_FZY.HPP
```

A.2.4. Fzy_unit.cpp

```
////////////////////////////////////////////////////////////////////////////
// -- 'FDB' object                                                        //
// -- Object used by the fuzzy algorithm                                  //
//                                                                        //
// -- FuzzysetDescriptorBlock class creation.  Allow to treat fuzzy sets.  //
// -- The fuzzy sets are stored as truth vectors with their associated     //
// -- bouding domains.                                                    //
////////////////////////////////////////////////////////////////////////////
#ifndef _FZY_UNIT.CPP
#define _FZY_UNIT.CPP

#include <iostream.h>
#include <stdio.h>
#include <conio.h>

#include "data_fzy.hpp"    // Constants for the fuzzy algorithm.


class far FDB
{
        bool        FDBempty;
                        // Is this a populated Fuzzyset?
        double      FDBdomain[2];
                        // Lo and Hi edges of the set
        float       FDBvector[VECMAX];
                        // The fuzzy set truth vector
        void FzyInterpVec(float*);
                        // Linear interpolation function
public :
        FDB (double, double);
                        // Constructor
        void FzyTriangleCurve(double, double, double);
                        // Creation a triangular fuzzet set.
```

```
        void FzyIntersection(FDB*);
                        // Takes the intersection between two fuzzy sets.
        double FzyDefuzzify(float *Grade);
                        // Returns the maximum value of a fuzzy set (with his degree of membership).
        ~FDB () {};
                        // Destructor.
};


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

FDB::FDB (double dom1=domain1, double dom2=domain2)
// Initialisation of the object
{
         int i;
         FDBempty       = TRUE;                                    // TRUE = 1
         FDBdomain[0]   = dom1;
         FDBdomain[1]   = dom2;
         for(i=0;i<VECMAX; i++) FDBvector[i]=0.0;   // FDBvector = (0.0,0.0,...,0.0)
}

////////////////////////////////////////////////////////////////////////////

void FDB::FzyInterpVec(float TempVector[VECMAX])
// This is a linear interpolation function.  A fuzzy set is constructed by
// linear interpolation between truth membership points across the domain.
{
        int    i,j,k,Pnt1,Pnt2;
        float  y,Seg1,Seg2;
        i=0;
        j=0;
                // This loop moves across the vector until it finds a slot that
                // does not contain a "-1".  We then linearly interpolate between
                // the previous boundary and this cell. The idea here is very
                // simple. We calculate "y" where is the proportion of the
                // distance from the anchor point to the current point (k).
                // We then multiply this with the linear distance between the two
                // points.
interpolate:
        j++;
        if(TempVector[j]==-1) goto interpolate;
        Pnt1=i+1;
        Pnt2=j;
        for(k=Pnt1;k<Pnt2+1;k++)
        {
                Seg1=k-i;
                Seg2=j-i;
                y=Seg1/Seg2;
                TempVector[k]= TempVector[i]+y*(TempVector[j]-TempVector[i]);
        }
        i=j;
        if(i>=VECMAX) return;
        goto interpolate;
}
```

```
////////////////////////////////////////////////////////////////////////////

void FDB::FzyTriangleCurve(double Left,double Center,double Right)
// Creation a triangular fuzzy set.
// This is a triangular around a central point.
{
   double  TrgPoint[3];     // Domain values
   float   TrgGrade[3];     // and correponding truth membership values

   TrgPoint[0]=Left;
   TrgGrade[0]=0.0;
   TrgPoint[1]=Center;
   TrgGrade[1]=1.0;
   TrgPoint[2]=Right;
   TrgGrade[2]=0.0;

   int     i,Cellpos;
   double  Hi,Lo,Range;
   float   tempVector[VECMAX];

   Hi=FDBdomain[1];
   Lo=FDBdomain[0];
   Range=Hi-Lo;

                // Initialisation of a temporary working vector to "-1"
   for (int j=0; j<VECMAX; j++)  tempVector[j]=-1.0;

                // Examination each of the domain values. For each domain value
                // (TrgPoint[i]) its cell position in the truth membership array
                // (tempVector[Cellpos]) is calculated.  This position is filled
                // with the correponding truth value (TrgGrade[i])
   for(i=0;i<3;i++)
   {
         Cellpos=(int)(((TrgPoint[i]-Lo)/Range)*VECMAX);
         tempVector[Cellpos]=TrgGrade[i];
   }

                // Initialisation of the extreme value of the temporary vector.
   if(tempVector[0]==-1) tempVector[0]=0.0;
   if(tempVector[VECMAX-1]==-1) tempVector[VECMAX-1]=0.0;

                // The linear interpolation routine is called to complete the fuzzy
                // set surface
   FzyInterpVec(tempVector);

                // Copy the tempVector in the FDBvector (the vector of the fuzzy set)
   for (i=0; i<VECMAX; i++)  FDBvector[i]=tempVector[i];

                // Adjust fuzzy set values
   FDBempty=FALSE;
}

////////////////////////////////////////////////////////////////////////////

void FDB::FzyIntersection(FDB *inFDBptr)
// Take the intersection between two fuzzy sets.  If the target fuzzy set is
```

```
// empty, then we just copy over the fuzzy set, otherwise we take the minimum
// of the two set across the domain.
{

                // empty fuzzy set
    if(FDBempty)
    {
          for (int i=0; i<VECMAX; i++)
                        FDBvector[i]=inFDBptr->FDBvector[i];
          FDBempty=FALSE;
    }

                // full fuzzy set
          else
    {
          for (int i=0; i<VECMAX; i++)
                        FDBvector[i]=min(FDBvector[i],inFDBptr->FDBvector[i]);
    }
}

////////////////////////////////////////////////////////////////////////////

double FDB::FzyDefuzzify(float *Grade)
// Return the maximum value of a fuzzy set, with his degree of membership.
// The maximum defuzzification method finds the point in the membership
// array with the maximum value.
{
          int       i,j,k;
          float     x;

          x=FDBvector[0];
          k=0;

                // Find the maximum vector and his position.
          for(i=0;i<VECMAX;i++)
                if(FDBvector[i]>x)
                {
                        x=FDBvector[i]; // maximum vector
                        k=i;            // his position
                }

                // Search the domain of the fuzzy set
          double Range=FDBdomain[1]-FDBdomain[0];
          double Scalar=FDBdomain[0]+(k*Range)/(VECMAX-1);

          *Grade=FDBvector[k];
          return(Scalar);
}

#endif   // _UNIT_FUZZY.CPP
```

A.2.5. Descript.cpp

```
////////////////////////////////////////////////////////////////////////////
```

```
// -- 'Descriptor' Object.                                                    //
// -- Contains the name of the descriptors and its weight.                    //
////////////////////////////////////////////////////////////////////////////////

#ifndef _DESCRIPT.CPP
#define _DESCRIPT.CPP

#include <string.h>

class Descriptors
{
        char * Name;
                        // Name of the descriptor.
        float  Weight;
                        // Weight of the descriptor
public :
        Descriptors ()
                { Weight = 1.0;                                                 }
                        // Constructor : Initialisation of 'Weight'
        void InitHeader (char * header)
                { Name = new char[strlen(header)+1];
                  stpcpy(Name,header);                          }
                        // Initialisation of 'Name'
        void ChangeWeight (float w)
                { Weight = Weight + w;                                          }
                        // Modify 'Weight'
        float GiveWeight ()
                { return (Weight);                                              }
                        // Return 'Weigth'
        char * GiveName ()
                { return(Name);                                                 }
                        // Return 'Name'
};

#endif   //_DESCRIPT.CPP
```

A.2.6. Graphobj.cpp

```
////////////////////////////////////////////////////////////////////////////////
// -- 'Draw', 'Rectangle' and 'Button' object.                                //
// -- Graphic objects for the construction of Rectangle and button on the     //
// -- screen.                                                                 //
////////////////////////////////////////////////////////////////////////////////
#ifndef _GRAPHOBJ.CPP
#define _GRAPHOBJ.CPP

#include <graphics.h>
#include <string.h>
#include <ctype.h>

class Draw
{

protected :
        short Color;
```

```
                // Color of the object
        short Thickness;
                // Thickness of the object
        short Textstyle;
                // Text style of the object
public :
        Draw (short c, short th, short ts)
                        { Color=c; Thickness=th; Textstyle=ts; }
                // Draw the object
        void ChangeColor(short nc)
                        { Color=nc; }
                // Change color of the object
};

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

class Rectangle : public Draw   // class derived from 'Draw' class
{
        int Left;
        int Top;
                // (left,top) is the upper left corner of the rectangle
        int Right;
        int Bottom;
                // (right,bottom) is its lower right corner
        short Type;
                // Type of the rectangle (0=BoxPrinc, 1=Box1,2,3,4)
public :
        Rectangle(short, short, short, int, int, int, int, short);
                // Constructor : Initialisation of variables
        void Show();
                // Show the rectangle on the screen
        int Inside(int, int);
                // if (int, int) is inside the rectangle, =1; otherwise = 0
        void WriteLeftTop(char *);
                // Write in the rectangle (left-top)
        void WriteCenter(char *);
                // Write in the rectangle (center)
};


////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

class Button : public Draw          // class derived from 'Draw' class
{
        int X;
        int Y;
                // (X,Y) is the center of the ellipse (=button)

        int Stangle;
        int Edangle;
                // the ellipse travels from strangle to endangle
        int Xradius;
                // horizontal axe of the ellipse
        int Yradius;
```

```
                // vertical axe of the ellipse
        char *Title;
                // Title of the button
public :
        Button (char*, short, short, short, int, int, int, int, int, int);
                // constructor : Initialisation of variables
        void Show ();
                // Show the button on the screen
        int Inside(int, int);
                // if (int, int) is inside the ellipse, =1 otherwise = 0
        void Write();
                // write the title in the middle of the button.
};

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

Rectangle::Rectangle(short c, short th, short ts, int l, int t, int r, int b, short ty) : Draw (c,th,ts)
// Constructor of Rectangle object : initialisation of variables
{
        Left=l;
        Top=t;
        Right=r;
        Bottom=b;
        Type=ty;    // 0=BoxPrinc, 1=else.
}

////////////////////////////////////////////////////////////////////////////////

void Rectangle::Show()
// Show the rectangle on the screen
{
        setcolor(Color);                    // initialise the color
        setlinestyle(0,1,Thickness);        // initialise the thickness
        rectangle(Left,Top,Right,Bottom);   // draw the rectangle
}

////////////////////////////////////////////////////////////////////////////////

int Rectangle::Inside(int x, int y)
// if (x,y) is inside the rectangle, return = 1; otherwise = 0
{
        return ((x>Left) && (x<Right) && (y>Top) && (y<Bottom));
}


////////////////////////////////////////////////////////////////////////////////

void Rectangle::WriteLeftTop(char * c)
// Write c in the rectangle (left-top)
{
   int x,y;
                // coords in current viewport
   int vport_start, vport_limit;
                // start & end limits of writing to viewport
   int start_flag;
```

```
                // is 1 if x is at start of a line, 0 if not
setviewport (Left+2, Top+2, Right-2, Bottom-2, 1);
                // set viewport
clearviewport();
                // clear viewport
setviewport (Left, Top, Right, Bottom, 1);
                // set viewport
settextstyle(Textstyle, 0,0);
                // set the current text characteristics
if (Type == 0)
{
        setusercharsize(2,3,1,2);      //
        vport_start = 7;               // For the Principal box
        vport_limit = 445;
}
else
{
        setusercharsize(1,1,1,1);      //
        vport_start = 8;               // For the Box1, box2, box3, box4
        vport_limit = 395;             //
}

x = vport_start;
y = 3;
                // set CP to "top left" (of viewport)
char letter[2];
                // used to advance CP by textwidth() * store letter
setcolor(14);
                // change color

for ( ; (*c != '\0'); *c++)
                // until end of c string
{
        char i;
        int letter_found = 0;
                        // = False
        int space_flag   = 0;
                        // = False
        for (i = ' '; ((i < '~') && (letter_found == 0))  ; i++)
                        // run through ascii characters until a match is found
        {
                        if (*c == i)
                                // if char of text = i
                        {
                                letter_found = 1;
                                        // True
                                letter[0] = i;
                                        // put in char for outtextxy() to use
                                letter[1] = '\0';
                                        // end it with null character
                        }
        }
        outtextxy(x, y, letter);
                        // write character to viewport
        if (isspace(letter[0]) && (x==vport_start))   space_flag=1;
                        // if first char written on the line has been a "space"
```

```
                        // don't inc. x coord
            if(!space_flag)
            {
                        if (x >= vport_limit)
                        {
                                x = vport_start; y += textheight(letter);
                        }
                        else x += textwidth(letter);
                                // to next space to write next char
            }
    }
    setviewport(0,0,638,477,1);
                // set general viewport
}

////////////////////////////////////////////////////////////////////////////////

void Rectangle::WriteCenter(char * ti)
// write ti in the rectangle (center)
{
        int posx, posy;
        settextstyle(Textstyle,0,0);
                // set text style
        setusercharsize(1,1,4,3);
                // set size of the font
        posx= (Left+(Right -Left)/2)-((textwidth(ti)) / 2);
        posy= (Top +(Bottom-Top )/2)-((textheight(ti)) / 2)-2;
                // compute the position of ti
        outtextxy(posx,posy,ti);
                // write ti
}

////////////////////////////////////////////////////////////////////////////////


Button::Button(char * t, short c, short th, short ts, int x, int y, int xr, int yr, int s=0, int e=360)
: Draw (c,th,ts)
// Constructor : Initialisation of the object
{
        X=x;
        Y=y;
        Xradius=xr;
        Yradius=yr;
        Stangle=s;
        Edangle=e;
        Title = new char[strlen(t)+1];
        strcpy(Title,t);
}

////////////////////////////////////////////////////////////////////////////////

void Button::Show()
// Draw the button on the screen
{
        setcolor(Color);
                // set the color
```

```
        setlinestyle(0,1,Thickness);
                // set tht thickness
        ellipse(X,Y,Stangle,Edangle,Xradius,Yradius);
                // draw the ellipse
        Write();
                // write the title in the ellipse
}

////////////////////////////////////////////////////////////////////////////

int Button::Inside(int coordx, int coordy)
// if (x,y) is inside the ellipse (=button) return = 1; otherwise = 0
{
        return( (X > (coordx-Xradius)) && (X < (coordx+Xradius))
                && (Y > (coordy-Yradius)) && (Y < (coordy+Yradius)) );
}

////////////////////////////////////////////////////////////////////////////

void Button::Write()
// write the title of the button on the middle of the ellipse (=button)
{
        int posx, posy;
        settextstyle(Textstyle,0,0);
                // set text style
        if (strlen(Title)>6) setusercharsize(1,2,1,2);
        else setusercharsize(2,3,1,2);
                // set the size of the text
        posx= (X - (textwidth (Title)) / 2);
        posy= (Y - (textheight(Title)) / 2)-3;
                // compute the position of the title
        outtextxy(posx,posy,Title);
                // write the title in the button

}


#endif  //      _GRAPHOBJ.CPP
```

A.2.7. Msmouse.hpp

```
////////////////////////////////////////////////////////////////////////////
// -- Header file for the mouse object                                    //
////////////////////////////////////////////////////////////////////////////
#ifndef H_MSMOUSE
#define H_MSMOUSE

        // defines graphics mouse cursor styles
struct HotSpotStruct { int X, Y; };

        // defines the structure for the mouse cursor
struct MouseCursor
{
           HotSpotStruct  HotSpot;
           unsigned ScreenMask[16];
```

```
            unsigned CursorMask[16];
};

        // Mouse cursor variables
extern const MouseCursor HandCursor;        // Hand
extern const MouseCursor MouthCursor;       // Mouth
extern const MouseCursor OMouthCursor;      // Open mouth
extern const MouseCursor CrossCursor;       // Cross
extern const MouseCursor IBeamCursor;       // I-Beam
extern const MouseCursor HourGlassCursor;   // Hour Glass

        // Mouse event codes
const unsigned Idle            = 0x0000;
                // Nothing
const unsigned MouseDown       = 0xff01;
                // A mouse button was pressed since previous call
const unsigned LMouseDown      = 0xff01;
                // The left mouse button was pressed since previous call
const unsigned RMouseDown      = 0xff02;
                // The rith mouse button was pressed since previous call
const unsigned MouseStillDown  = 0xff04;
                // One of the mouse buttons is still pressed
const unsigned LMouseStillDown = 0xff04;
                // The left mouse button is still pressed
const unsigned RMouseStillDown = 0xff08;
                // The right mouse button is still pressed
const unsigned MouseUp         = 0xff10;
                // One of the mouse buttons was just released
const unsigned LMouseUp        = 0xff10;
                // The left mouse button was just released
const unsigned RMouseUp        = 0xff20;
                // The right mouse button was just released
const unsigned MouseEnter      = 0xff40;
                // The mouse cursor has entered a specified region
const unsigned MouseLeave      = 0xff80;
                // The mouse cursor has left a specified region
const unsigned MouseWithin     = 0xffc0;
                // The mouse is within a specified region

        // Mouse Button Masks
const unsigned LeftButton      = 0x0001;
const unsigned RightButton     = 0x0002;

        // The video mode that the mouse is running under
enum VideoModeType {TextScrn, LowResGr, HerculesGr, Graphics} ;

        // The mouse class
class MouseObject {

protected:
   int      OldX, OldY;
                // Used solely by Moved to keep position
   char     OK;
                // True if initialised
   char     MouseOff;
                // True if mouse is disabled (Default)
```

```
    char      LowRes;
                  // True if in 320x200 graphics mode
    char      TextMode;
                  // True if in Text Mode

public:

    int       X, Y, Dx, Dy;
                  // Keeps track of the mouses movement
    MouseObject(void);
    void      Setup(VideoModeType Videomode);
    int       DriverExists(void);
    int       SetUpOK(void);
    void      Hide(void);
    void      Show(void);
    unsigned  Status(int &Mx, int &My);
    unsigned  ButtonStatus(void);
    int       PressCnt(unsigned ButtonMask);
    int       ReleaseCnt(unsigned ButtonMask);
    unsigned  Event(int &Mx, int &My);
    unsigned  WaitForAnyEvent(int &Mx, int &My);
    void      WaitForEvent(unsigned E, int &Mx, int &My);
    int       Moved(void);
    void      Move(int Mx, int My);
    void      TurnOn(void);
    void      TurnOff(void);
    int       Operating(void);
    void      SetGCursor(const MouseCursor &NewCursor);
};

extern MouseObject Mouse;

#endif
```

A.2.8. Mcursor.cpp

```
////////////////////////////////////////////////////////////////////////////////
// -- Definition of the pointer shapes                                        //
////////////////////////////////////////////////////////////////////////////////
#ifndef _MCURSOR.CPP
#define _MCURSOR.CPP

#include "msmouse.hpp"

        // Hand cursor
const MouseCursor HandCursor = {
    { 4, 0 },                                  // Hot spot at tip of pointing finger
    { 0xF3FF, 0xE1FF, 0xE1FF, 0xE1FF,          // Screen Mask
          0xE001, 0xE000, 0xE000, 0xE000,
          0x8000, 0x0000, 0x0000, 0x0000,
          0x0000, 0x0000, 0x8001, 0xC003 },
    { 0x0000, 0x0C00, 0x0C00, 0x0C00,          // Cursor Mask
          0x0C00, 0x0DB6, 0x0DB6, 0x0DB6,
          0x0DB6, 0x6FFE, 0x6FFE, 0x6FFE,
          0x7FFE, 0x7FFE, 0x3FFC, 0x0000 }
```

```
);


        // Mouth cursor
const MouseCursor MouthCursor = {
        { 8, 8 },                               // Hot spot in middle
        { 0xffff, 0xffff, 0xffff, 0xf3cf,   // Screen Mask
          0xe187, 0xc003, 0x8001, 0x0000,
          0x8001, 0xc003, 0xe007, 0xf00f
          0xffff, 0xffff, 0xffff, 0xffff },
        { 0x0000, 0x0000, 0x0000, 0x0000,   // Cursor Mask
          0x0000, 0x0c30, 0x1ff8, 0x07e0,
          0x1818, 0x0ff0, 0x07e0, 0x0000,
          0x0000, 0x0000  0x0000, 0x0000 }
);


        // Mouth open cursor
const MouseCursor OMouthCursor = {
        { 8, 8 },                               // Hot spot in middle
        { 0xfbdf, 0xf18f, 0xe007, 0xc003,   // Screen Mask
          0x8001, 0x0ff0, 0x1ff8, 0x3ffc,
          0x0ff0, 0x83c1, 0xc003, 0xe007,
          0xf00f, 0xf81f, 0xffff, 0xffff },
        { 0x0000, 0x0420, 0x0e70, 0x1ff8,   // Cursor Mask
          0x300c, 0x6006, 0x4002, 0x0000,
          0x4002, 0x300c, 0x1c38, 0x0ff0,
          0x07e0, 0x0000, 0x0000, 0x0000 }
);


        // Cross cursor
const MouseCursor CrossCursor = {
        { 8, 8 },                               // Hot spot in middle
        { 0xffff, 0xffff, 0xffff, 0xe7e7,   // Screen Mask
          0xe3c7, 0xf18f, 0xf81f, 0xfc3f,
          0xfc3f, 0xf81f, 0xf18f, 0xe3c7,
          0xe7e7, 0xffff, 0xffff, 0xffff },
        { 0x0000, 0x0000, 0x0000, 0x1818,   // Cursor Mask
          0x1c38, 0x0e70, 0x07e0, 0x03c0,
          0x03c0, 0x07e0, 0x0e70, 0x1c38,
          0x1818, 0x0000, 0x0000, 0x0000 }
);


        // I-Beam cursor
const MouseCursor IBeamCursor = {
        { 8, 8 },                               // Hot spot in middle
        { 0xf11f, 0xfeff, 0xfeff, 0xfeff,   // Screen Mask
          0xfeff, 0xfeff, 0xfeff, 0xfeff,
          0xfeff, 0xfeff, 0xfeff, 0xfeff,
          0xfeff, 0xfeff, 0xfeff, 0xf11f },
        { 0x0ee0, 0x0100, 0x0100, 0x0100,   // Cursor Mask
          0x0100, 0x0100, 0x0100, 0x0100,
          0x0100, 0x0100, 0x0100, 0x0100,
          0x0100, 0x0100, 0x0100, 0x0ee0 }
```

```
};

        // Hour glass cursor
const MouseCursor HourGlassCursor = {
        { 8, 8 },                       // Hot spot in middle
        { 0x8003, 0x8003, 0xc007, 0xc007    // Screen Mask
          0xc007, 0xe00f, 0xf01f, 0xf83f,
          0xf83f, 0xf01f, 0xe00f, 0xc007,
          0xc007, 0xc007, 0x8003, 0x8003 },
        { 0x7ffc, 0x7ffc, 0x3018, 0x3018,   // Cursor Mask
          0x3058, 0x1ab0, 0x0d60, 0x06c0,
          0x06c0, 0x0d60, 0x1830, 0x3118,
          0x3298, 0x3558, 0x7ffc, 0x7ffc}
};


#endif   // _MCURSOR.CPP
```

A.2.9. Msmouse.cpp

```
//////////////////////////////////////////////////////////////////////////////
// -- Mouse object                                                          //
//////////////////////////////////////////////////////////////////////////////
#ifndef _MSMOUSE.CPP
#define _MSMOUSE.CPP

#include <dos.h>
#include <stdlib.h>

#include "msmouse.hpp"

const int MsCall  = 0x33;
const int Iret    = 0xcf;
const int False   = 0;
const int True    = 1;


        // ------------------------------------------------------ //


MouseObject::MouseObject(void)
//Initializes mouse to a known state. Mouse is not turned on yet
{
  OK = False;
  MouseOff = True;
}

        // ------------------------------------------------------ //

int MouseObject::DriverExists(void)
// Returns true if a mouse driver is installed. This function maked sure
// the interrupt vector location serviced by the mouse is pointing to
// something-hopefully the mouse driver.
```

```
{
  void far *address;
  // Look for NULL address or Iret instruction
  address = getvect(MsCall);
  return (address != NULL) && (*(unsigned char far *)address != Iret);
}

        // ------------------------------------------------------ //

void MouseObject::Setup(VideoModeType VideoMode)
// Initializes the mouse object by verifying that the mouse driver exists
// that the mouse responds to its initialization function, by moving the
// mouse to the top oleft of the screen, and by setting various internal
// variables.Call the funcion SetupOK after calling Init to find out if the
// mouse initialization was successful.
// The video mode parameter specifies which videc mode the mouse is being
// used under.
{
  /*struct?*/REGS regs;

  OK = DriverExists();
  if (OK)  {  // Mouse present, but will it reset?
  // Fix up hercules mode for page 0. Use 5 for page 1.
        if (VideoMode == HerculesGr)  *(char far *)MK_FP(0x0040, 0x0049) = 6;
        regs.x.ax = 0;
        int86(MsCall, &regs, &regs);
        if (regs.x.ax == 0) OK = False;
  }
  if (!OK)  {   // Mouse initialization failed. Set
        TurnOff(); // the mouse state off and return.
        return;
  }
  TurnOn();   // set the mouse state on
  if (VideoMode == TextScrn)  TextMode = True; else TextMode = False;
  if (VideoMode == LowResGr)  LowRes = True; else LowRes = False;

  OldX = 0; OldY = 0;  // Initialize various variables
  X = 0; Y = 0;
  Dx = 0;  Dy = 0;
  Move(0,0);  // Set the mouse to topm left of screen
}

        // ------------------------------------------------------ //

int MouseObject::SetUpOK(void)
// Returns true only if the mouse initialization was successfule
{
  return OK;
}

        // ------------------------------------------------------ //

void MouseObject::Hide(void)
// Hide the mouse cursor. Call this function to remove the mouse
// cursor from the screen
{
```

```
  /*struct?*/ REGS regs;
  if (!Operating()) return;
  regs.x.ax = 2;
  int86(MsCall, &regs, &regs);
}

        // ------------------------------------------------------ //

void MouseObject::Show(void)
// Displays the mouse cursor
{
   /*struct?*/ REGS regs;
   if (!Operating()) return;
   regs.x.ax = 1;
   int86(MsCall, &regs, &regs);
}

        // ------------------------------------------------------ //

unsigned MouseObject::Status(int &Mx, int &My)
// This general function returns the location of the mouse in Mx,My
// and the current status of the buttons in the function name
{
  /*struct?*/REGS regs;

  if (!Operating()) { Mx = 0; My = 0; return 0; }
  regs.x.ax = 3;
  int86(MsCall, &regs, &regs);
  Mx = regs.x.cx;  My = regs.x.dx;
  if (TextMode)  {
        Mx >>=3;  // Adjust for text coordinated
        My >>=3;
  }
  if (LowRes)  Mx >>=1;  // Adjust for 320x200 coords
  return regs.x.bx;
}

        // ------------------------------------------------------ //

unsigned MouseObject::ButtonStatus(void)
// Returns the status of the mouse button
{
   int Mx, My;
   if (!Operating()) return 0; else return Status(Mx,My);
}

        // ------------------------------------------------------ //

int MouseObject::PressCnt(unsigned ButtonMask)
// Returns the number of times the button has been pressed since last time called
{
   REGS regs;
   if (!Operating()) return 0;
   regs.x.ax = 5;
   regs.x.bx = ButtonMask >> 1; // Button selector
   int86(MsCall, &regs, &regs);
```

```
    return regs.x.bx;
}

        // ------------------------------------------------------ //

int MouseObject::ReleaseCnt(unsigned ButtonMask)
// Returns no of timed button has been released since last time called
{
    REGS regs;
    if (!Operating()) return 0;
    regs.x.ax = 6;
    regs.x.bx = ButtonMask >> 1;  // Button selector
    int86(MsCall, &regs, &regs);
    return regs.x.bx;
}

        // ------------------------------------------------------ //

unsigned MouseObject::Event(int &Mx, int &My)
// Gets the last mouse event. The left button has priority over the right button
{
    unsigned E;

    if (!Operating())  { Mx = 0; My = 0; return Idle; }
    // Get current status of buttons
    E = Status(Mx, My);
    if (E == 0) {
            // No mouse button down but maybe there was a button press that was
            // missed. If not, check to see whether a button release was missed
            // Favour the left mouse button
            if (PressCnt(LeftButton) > 0) E = LMouseDown;
            else if (PressCnt(RightButton) > 0) E = RMouseDown;
            // Maybe left one was just released
            else if (ReleaseCnt(LeftButton) > 0) E = LMouseUp;
            // Maybe right one wa just released
            else if (ReleaseCnt(RightButton) > 0) E = RMouseUp;
    }
    else  {   // a mouse button is down
            // was the left button already down?
            if (E & LeftButton)  {
            // Not already down
            if (PressCnt(LeftButton) > 0)
                  E = LMouseDown;            // Must have just been pressed
                  else E = LMouseStillDown;  // Al;ready down
            }
            else if (PressCnt(RightButton) > 0)
            E = RMouseDown;             // Must have just been pressed
            else E = RMouseStillDown;   // Already down
    }
    return E;  // Return the mouse event code
}

        // ------------------------------------------------------ //

unsigned MouseObject::WaitForAnyEvent(int &Mx, int &My)
// Waits for a mouse event to occur and return its code
```

```
{
   unsigned E;

   if (!Operating())  { Mx = 0; My = 0; return Idle; }
   do {
         E = Event(Mx,My);
   } while (E == Idle);
   return E;
}

         // ------------------------------------------------------ //

void MouseObject::WaitForEvent(unsigned E, int &Mx, int &My)
// Waits for the event E to occur. Return its mouse coordinates
{
   unsigned Etry;
   if (!Operating())  { Mx = 0; My = 0; return; }
   do {
         Etry = Event(Mx,My);
   } while (Etry != E);
}

         // ------------------------------------------------------ //

int MouseObject::Moved(void)
// Tests to see if the mouse has moved since the last ime this function called
{
   if (!Operating())  return False;
   OldX = X;  OldY = Y;
   Status(X,Y);
   Dx = X - OldX;  Dy = Y - OldY;
   return (Dx != 0) || (Dy != 0);
}

         // ------------------------------------------------------ //

void MouseObject::Move(int Mx, int My)
// MOves the mouse cursor
{
   REGS regs;
   if (!Operating()) return;
   regs.x.ax = 4;
   regs.x.cx = Mx;
   regs.x.dx = My;
   if (TextMode)   {   // Adjust for text coordinated
         regs.x.cx <<=3;
         regs.x.dx <<=3;
   }
   if (LowRes) regs.x.cx <<=1; // Adjust for 320x200 coords
   int86(MsCall, &regs, &regs);
}

         // ------------------------------------------------------ //

void MouseObject::TurnOn(void)
// Enables the mouse code
```

```
{
   if (OK && MouseOff)  {
            MouseOff = False;
            Show();
   }
}

         // ------------------------------------------------------ //

void MouseObject::TurnOff(void)
// Disables the mouse code. This is usefule when you don't want to use
// the mouse, but the code already has mouse calls in it.
{
   if (OK && !MouseOff)  {
            Hide();
            MouseOff = True;
   }
}

         // ------------------------------------------------------ //

int MouseObject::Operating(void)
// Returns a boolean flag that is true only if the mouse object has been
// enabled. This is the default state
{
  return !MouseOff;
}

         // ------------------------------------------------------ //

void MouseObject::SetGCursor(const MouseCursor &NewCursor)
// Sets the graphics mouse cursor to the type specified
{
   REGS regs;
   SREGS sregs;

   if (!Operating()) return;
   regs.x.ax = 9;
   regs.x.bx = NewCursor.HotSpot.X;
   regs.x.cx = NewCursor.HotSpot.Y;
   regs.x.dx = FP_OFF(NewCursor.ScreenMask);
   sregs.es  = FP_SEG(NewCursor.ScreenMask);
   int86x(MsCall, &regs, &regs, &sregs);
}


#endif   // _MSMOUSE.CPP
```

A.3. Module files

A.3.1. Unloadin.cpp

```
////////////////////////////////////////////////////////////////////////////
```

```
// -- Functions of the Unloading module.                                        //
////////////////////////////////////////////////////////////////////////////////
#ifndef _UNLOADING.CPP
#define _UNLOADING.CPP

#include <constream.h>
#include <time.h>
#include <fstream.h>
#include <string.h>
#include <stdlib.h>


void Used_Fuzzy(int nb);
        // utilisation of a function of the Speechbox module.


////////////////////////////////////////////////////////////////////////////////


void Database_file()
// Function whom fill the database array (object) with the data in
// the 'database.in' file.
{
        clrscr();
        int item=0;
        int i;
        int temp;
        char ph[Max_conversation_item+1];
        char de[Max_length_descriptors+1];
        fstream file("database.in",ios::in);
                // pointer on the 'database.in' file

        if (file==NULL)
                // if 'database.in' file don't exist
        {
                cout << "\Error : 'Database.in' file inexisting !";
                exit(0);
        }


// read the number of subject, purpose and person descritors.
        file >> Nb_subject;
        file >> Nb_purpose;
        file >> Nb_person;

// read the names of descriptors (subject, purpose, person)
        for (i=0;i<Nb_subject;i++)
        {       file >> de;
                Subject_descriptor[i].InitHeader(de);
        }
        for (i=0;i<Nb_purpose;i++)
        {       file >> de;
                Purpose_descriptor[i].InitHeader(de);
        }
        for (i=0;i<Nb_person;i++)
        {       file >> de;
```

```
                Person_descriptor[i].InitHeader(de);
        }
        file >> de;

// for each conversation item record.
        while ((!(file.eof())) && (item < Max_item))
        {

        // read the value of the descriptors.
                for (i=0;i<Nb_subject;i++)
                {       file >> temp;
                        Database[item].InitSubject(temp,i);
                }
                for (i=0;i<Nb_purpose;i++)
                {       file >> temp;
                        Database[item].InitPurpose(temp,i);
                }
                for (i=0;i<Nb_person;i++)
                {       file >> temp;
                        Database[item].InitPerson(temp,i);
                }

        // read the frequency.
                file >> temp;
                Database[item].InitFrequency(temp);

        // read the conversation item.
                file.get (ph, Max_conversation_item);
                Database[item].InitConversationItem(ph);
                item++;
        }

// Initialisation of the global variable (Nb_item)
        Nb_item=item;
}

////////////////////////////////////////////////////////////////////////////////

void Decrypt_descriptors(char d[Max_length_descriptors])
// Add the 'weight_diary' constant, at the default value (=1) of
// the weight's descriptors.
// Look for if the name of the descriptor is egal to d, and if yes, change
// the weight of the descriptor.
//
// Function uses by the 'diary_file' function.
{
        int find=0;
        for (int i=0;i<Nb_subject;i++)
                // seek in the Subjet_descriptor
        {
                if (strcmp(d,Subject_descriptor[i].GiveName())==0)
                {
                        Subject_descriptor[i].ChangeWeight(Weight_diary);
                        find=1;
                }
        }
```

```
        for (i=0;i<Nb_purpose;i++)
                // seek in the Purpose_descriptor
        {
                if (strcmp(d,Purpose_descriptor[i].GiveName())==0)
                {
                        Purpose_descriptor[i].ChangeWeight(Weight_diary);
                        find=1;
                }
        }
        for (i=0;i<Nb_person;i++)
                // seek in the Person_descriptor
        {
                if (strcmp(d,Person_descriptor[i].GiveName())==0)
                {
                        Person_descriptor[i].ChangeWeight(Weight_diary);
                        find=1;
                }
        }
        if (find==0)
                // if not found
        {
                cout << "\nWarning : Descriptors '" << d << "' not the same in the 2 input files !";
                getch();
        }
}

////////////////////////////////////////////////////////////////////////////////

void Diary_file()
// Uses the file 'diary.in'.
// Add more weight to descriptors if they are in the diary file (and if they
// are in the good place).
{
        fstream file("diary.in",ios::in);
                // pointer on the 'diary.in' file

        if (file==NULL)
                // if 'diary.in' file don't exist
        {
                cout << "\nWarning : 'Diary.in' file don't exist !!";
                getch();
        }
        else
        {
                time_t timer;
                struct tm *tblock;
                timer=time(NULL);
                tblock = localtime(&timer);
                        // variable contain a struct with the date and the time.

                int ok=0;
                char day[Max_length_descriptors];

// Seek the good day in the diary_file.
                while (!ok)
                {
```

```
                        file >> day;
                        if (((strcmp(day,"monday")==0) || (strcmp(day,"Monday")==0))
                                && (tblock->tm_wday==1)) ok=1;
                        if (((strcmp(day,"tuesday")==0) || (strcmp(day,"Tuesday")==0))
                                && (tblock->tm_wday==2)) ok=1;
                        if (((strcmp(day,"wednesday")==0) || (strcmp(day,"Wednesday")==0))
                                && (tblock->tm_wday==3)) ok=1;
                        if (((strcmp(day,"thursday")==0) || (strcmp(day,"Thursday")==0))
                                && (tblock->tm_wday==4)) ok=1;
                        if (((strcmp(day,"friday")==0) || (strcmp(day,"Friday")==0))
                                && (tblock->tm_wday==5)) ok=1;
                        if (((strcmp(day,"saturday")==0) || (strcmp(day,"Saturday")==0))
                                && (tblock->tm_wday==6)) ok=1;
                        if (((strcmp(day,"sunday")==0) || (strcmp(day,"Sunday")==0))
                                && (tblock->tm_wday==0)) ok=1;
                        if (file.eof())
                                // if not found
                        {
                                cout <<"\nError : Structure not good in the file 'Diary.in' !!";
                                exit(0);
                        }
                }
                ok=0;

// Creation of the variable 'ghour'.
                int hour;
                int ghour;
                ghour = tblock->tm_hour *100;

// Seek the good hour in the diary file.
                while (!ok)
                {

                        file >> day;
                        hour=atoi(day);
                        if (hour==ghour) ok=1;
                        if (file.eof()) ok=1;
                }

// Find the names of the descriptors.
                if (!(file.eof()))
                {
                        file>>day;
                        while (atoi(day)==0)
                        {
                                Decrypt_descriptors(day); // change the weight of descriptors.
                                file>>day;
                        }
                        ok=0;

// Creation of the variable 'r'.
                        char r[10];
                        char d[3];
                        char m[3];
                        char y[3];
                        char s[2];
```

```
                        s[0]='/';
                        s[1]='\0';
                        int mon=(tblock->tm_mon) +1;
                        itoa(tblock->tm_mday,d,10);
                        itoa(mon,m,10);
                        itoa(tblock->tm_year,y,10);
                        strcpy(r,d);
                        strcat(r,s);
                        strcat(r,m);
                        strcat(r,s);
                        strcat(r,y);

// Find the good date in the diary file (if day=r)
                        while (!ok)
                        {
                                file >> day;
                                if (strcmp(day,r)==0)
                                {
                                        ok=1;
                                        file >> day;

                                // Find the names of the descriptors.
                                        while ((atoi(day)==0) && (!(file.eof())))
                                        {
                                                Decrypt_descriptors(day);
                                                        // change the weight of descriptors.
                                                if (!(file.eof())) file>>day;
                                        }
                                }

                                if (file.eof()) ok=1;
                        }
                }
        }
}

////////////////////////////////////////////////////////////////////////////

void SetUp()
// Use the 'setup.in' file to initialise : COM and Synthesiser variables
{
        fstream file("setup.in",ios::in);
                // pointer on the 'setup.in' file

        char set[10];
        int s,nb;

        if (file==NULL)
                // if file don't exist
        {
                cout << "\nWarning : 'Setup.in' file don't exist !!";
                getch();
        }
        else
        {
                while (!(file.eof()))
```

```
                        // while not the end of the file
                {
                        file >> set;
                        s=0;
                        if (strcmp(set,"COM")==0) s=1;
                        if (strcmp(set,"SYNTH")==0) s=2;
                        switch (s)
                        {

                                // if set=COM
                                case 1 : file >> nb;
                                                if (nb==1) COM=0;
                                                if (nb==2) COM=1;
                                                break;

                                // if set=SYNTH
                                case 2 : file >> set;
                        if ((strcmp(set,"ON" )==0) || (strcmp(set,"on")==0)) Synthesiser = 1;
                        if ((strcmp(set,"OFF")==0) || (strcmp(set,"off")==0)) Synthesiser = 0;
                                                break;
                        }
                }
        }

}

////////////////////////////////////////////////////////////////////////////////

void Initialisation()
// Make the other initialisation of global variables and graphics.
{
        Freeze=0;
        Back_down=0;
        BackDownFuzzy();
        BoxPrinc=-1;
        Box1=-1;
        Box2=-1;
        Box3=-1;
        Box4=-1;
                                        // Initialisation of global variables

        apollo(Database[0].GiveConversation(),COM);
                                        // Speak the first item conversatin.

        Used_Fuzzy(0);    // First research of the clotest item to the first
                                        // item conversation.


}


#endif   // _UNLOADIN.CPP
```

A.3.2. Spechbox.cpp

```
////////////////////////////////////////////////////////////////////////////////
// -- Functions of the modules Speechbox                                      //
// -- Treatment of 5 box and buttons 'more' '<' '>' and freeze.               //
// -- Fuzzy algorithm                                                         //
////////////////////////////////////////////////////////////////////////////////
#ifndef _SPECHBOX.CPP
#define _SPECHBOX.CPP


#include <constream.h>
#include <stdlib.h>


// structure use in the fuzzy algorithm
struct far_st
{
        FDB psub[Max_subject];
        FDB ppur[Max_purpose];
        FDB pper[Max_person];
};



void Used_Fuzzy (int nb);                              //
                                                       // See after this
                                                       //
float Degree_Membership (st_far * nn, int nouv);       //

////////////////////////////////////////////////////////////////////////////////

void Freezer()
// Treatment of the <freeze> button.
{
        if (Freeze==0)
                // if the screen is not freezing
        {
                Freeze=1;
                        // Update the variable Freeze

                Mouse.Hide();
                FreezeG.ChangeColor(1);
                FreezeG.Show();
                Unfreeze.ChangeColor(10);
                Unfreeze.Show();
                Boites_freezes();
                Mouse.Show();
                        // freezing of the screen

                Conversation_file("FREEZE BUTTON","");
                        // Update the 'conversa.out' file

        }
        else
```

```
                // if the screen is freezing
        {
                Freeze=0;
                        // Update the variable Freeze.

                Mouse.Hide();
                Unfreeze.ChangeColor(1);
                Unfreeze.Show();
                FreezeG.ChangeColor(10);
                FreezeG.Show();
                Boites_normales();
                Mouse.Show();
                        // unfreezing of the screen

                Conversation_file("UNFREEZE BUTTON","");
                        // Update the 'conversa.out' file

        }
}

////////////////////////////////////////////////////////////////////////////////

void Find_Clotest (int box)
// Function actived if the user clicks on one of the 5 speech boxes.
// Launch the search on the database to find the closest items
{
        if (box==0)
        // = Principal box
        {
                apollo(Database[BoxPrinc].GiveConversation(),COM);
                        // Speech the conversation item of the Principal box.

                Conversation_file("BOX 0 : ",Database[BoxPrinc].GiveConversation());
                        // Update the 'conversa.out' file
        }

        if (box==1)
        // Box 1
        {
                apollo(Database[Box1].GiveConversation(),COM);
                        // Speech the conversation item of the box 1.

                Conversation_file("BOX 1 : ",Database[Box1].GiveConversation());
                        // Udate the 'conversa.out' file

                if ((Database[Box1].GiveUtilisation()==-1))
                        Database[Box1].ChangeUtilisation(1);
                        // Update the variable 'Utilisation' in the database object.

                if (Freeze==0) Used_Fuzzy(Box1);
                        // Seek the closest item to the box 1 (if not freeze).
        }

        if (box==2)       // idem with box2
        {
                apollo(Database[Box2].GiveConversation(),COM);
```

```
                Conversation_file("BOX 2 : ",Database[Box2].GiveConversation());
                if ((Database[Box2].GiveUtilisation()==-1))
                                Database[Box2].ChangeUtilisation(1);
                if (Freeze==0) Used_Fuzzy(Box2);
        }

        if (box==3)       // idem with box3
        {
                apollo(Database[Box3].GiveConversation(),COM);
                Conversation_file("BOX 3 : ",Database[Box3].GiveConversation());
                if ((Database[Box3].GiveUtilisation()==-1))
                                Database[Box3].ChangeUtilisation(1);
                if (Freeze==0) Used_Fuzzy(Box3);
        }

        if (box==4)       // idem with box4
        {
                apollo(Database[Box4].GiveConversation(),COM);
                Conversation_file("BOX 4 : ",Database[Box4].GiveConversation());
                if ((Database[Box4].GiveUtilisation()==-1))
                                Database[Box4].ChangeUtilisation(1);
                if (Freeze==0) Used_Fuzzy(Box4);
        }
}

////////////////////////////////////////////////////////////////////////////////

void More_button()
// Treatment of the <More> button.

{
        Conversation_file("MORE BUTTON","");
                        // Update the 'conversa.out' file

        Used_Fuzzy(BoxPrinc);
                        // Seek closest items to the principal box.
}

////////////////////////////////////////////////////////////////////////////////

void Backtrack_up()
// Treatment of the '>' button.

{
        int P,b1,b2,b3,b4;

// Find the next element.
        Backtrack.GiveNext (P,b1,b2,b3,b4);

// Update the screen and the global variables.
        BoxPrinc=P;
        BoxPrincG.WriteLeftTop(Database[P].GiveConversation());
        Box1=b1;
        Box1G.WriteLeftTop(Database[b1].GiveConversation());
        Box2=b2;
        Box2G.WriteLeftTop(Database[b2].GiveConversation());
```

```
        Box3=b3;
        Box3G.WriteLeftTop(Database[b3].GiveConversation());
        Box4=b4;
        Box4G.WriteLeftTop(Database[b4].GiveConversation());

// Update the 'conversa.out' variable
        Conversation_file("> BUTTON","");

// Udapte the '<' button and '>' button, if necessary
        Back_down=1;
        BackDownNet();
        if (Backtrack.Last())
        {
                Back_up=0;
                BackUpFuzzy();
        }

}

////////////////////////////////////////////////////////////////////////////

void Backtrack_down()
// Treatment of the '<' button.

{
        int P,b1,b2,b3,b4;

// Find the precedent element.
        Backtrack.GivePrec (P,b1,b2,b3,b4);

// Update the screen and the global variables.
        BoxPrinc=P;
        BoxPrincG.WriteLeftTop(Database[P].GiveConversation());
        Box1=b1;
        Box1G.WriteLeftTop(Database[b1].GiveConversation());
        Box2=b2;
        Box2G.WriteLeftTop(Database[b2].GiveConversation());
        Box3=b3;
        Box3G.WriteLeftTop(Database[b3].GiveConversation());
        Box4=b4;
        Box4G.WriteLeftTop(Database[b4].GiveConversation());

// Update the 'conversa.out' file
        Conversation_file("< BUTTON","");

// Update the '<' and '>' buttons, if necessary
        Back_up=1;
        BackUpNet();
        if (Backtrack.First())
        {
                Back_down=0;
                BackDownFuzzy();
        }

}
```

```
////////////////////////////////////////////////////////////////////////////////

void Used_Fuzzy(int nb)
// Find the 4 clotest item of nb (=index in the database array).
//
// - Construction (= structure 'nouv') of triangular fuzzy sets for all
// descriptors (subject,purpose and person) for the item 'database[nb]'
//
// - Call the function 'Degree_Membership' for each item in the database to
// find 4 closest item to 'database[nb]'.

{
        Mouse.SetGCursor(HourGlassCursor);
                // Update the shape pointer

        int n1=nb;
        int n2=nb;
        int n3=nb;
        int n4=nb;
                // will contain index (in the 'database' array) of the four
                // closest items to 'database[nb]'.

        float np1=0.0;
        float np2=0.0;
        float np3=0.0;
        float np4=0.0;
                // will contain the difference between 'database[nb]' (the item in the
                // principal box) and 'database[l] (with l = (n1,n2,n3,n4)), the four
                // closest items to 'database[nb]'

        if (BoxPrinc!=-1)
                // if 'used_fuzzy' isn't called during the initialisation (= after the
initialisation).
        {
                Database[BoxPrinc].ChangeUsed(0);
                Database[Box1].ChangeUsed(0);
                Database[Box2].ChangeUsed(0);
                Database[Box3].ChangeUsed(0);
                Database[Box4].ChangeUsed(0);
                        // Update of the Used of each item display on the screen

                Back_down=1;
                BackDownNet();
                        // Update the '<' button.

        }

        float profile;
                // variable who computes the value of one element.

        static st far * nouv;
        nouv = new st;
                // pointer on the structure 'st'
                // nouv will contain all triangular fuzzy sets (for each subject,
                // purpose and person descriptor) of the text item display on the principal
                // box (nb = index in 'Database' array of this item)
```

```
        if (nouv==NULL)
                // if there are not enough of memory for the allocation
        {
                closegraph();
                cout << "\nError : Not enough of memory !";
                getch();
                exit(1);
        }

        int ca, da, ga;

// Construction of the structure 'nouv'
        for (int i=0;i<Nb_subject;i++)
                // for each subject descriptor
        {
                ca = Database[nb].GiveSubject(i);
                        // ca = subject[i] descriptor of Database[nb]
                        // ca is the center of the triangular fuzzy set of the subject[i] descriptor

                ga = ca - largeur; if (ga<-1) ga=-1;
                // ga is the left limit of the triangular fuzzy set of the subject[i] descriptor

                da = ca + largeur; if (da>21) da=21;
                // da is the right limit of the triangular fuzzy set of the subject[i] descriptor

                nouv->psub[i].FzyTriangleCurve(ga,ca,da);
                // creation of the triangular fuzzy set for the subject[i] descriptor
                // of the text item display on principal box
        }

        for (i=0;i<Nb_purpose;i++)
                // idem for each purpose descriptor
        {
                ca = Database[nb].GivePurpose(i);
                ga = ca - largeur; if (ga<-1) ga=-1;
                da = ca + largeur; if (da>21) da=21;
                nouv->ppur[i].FzyTriangleCurve(ga,ca,da);
        }

        for (i=0;i<Nb_person;i++)
                // idem for each person descriptor
        {
                ca = Database[nb].GivePerson(i);
                ga = ca - largeur; if (ga<-1) ga=-1;
                da = ca + largeur; if (da>21) da=21;
                nouv->pper[i].FzyTriangleCurve(ga,ca,da);
        }



// for each item in the database variable.
        for (i=0;i<Nb_item;i++)
        {
        // except the nb item.
          if (i!=nb)
```

```
        {
                profile = Database[i].GiveUsed();
                        // value of the recent use.

                profile += Degree_Membership (nouv,i);
                        // compute the difference between the 'database[nb]' (->nouv) and
                        // the 'database[i]'
                        // (=fuzzy algorithm).

                        // Compute of 4 maximum values.
                if (profile>np1)
                {
                        n4=n3;np4=np3;
                        n3=n2;np3=np2;
                        n2=n1;np2=np1;
                        n1=i; np1=profile;
                }
                else if (profile>np2)
                {
                        n4=n3;np4=np3;
                        n3=n2;np3=np2;
                        n2=i; np2=profile;
                }
                else if (profile>np3)
                {
                        n4=n3;np4=np3;
                        n3=i; np3=profile;
                }
                else if (profile>np4)
                {
                        n4=i; np4=profile;
                }
        }

                // Modify the recently used.
          Database[i].ModifyUsed(Fzy_augmentation);
          if (Database[i].GiveUsed() > Fzy_recently_used)
                        Database[i].ChangeUsed(Fzy_recently_used);

      }

      delete far nouv;

// Update the global variables, the screen, and the 'utilisation' variable in
// the database object.
      Mouse.Hide();
      Back_up=0;
      BackUpFuzzy();
      BoxPrinc=nb;
      BoxPrincG.WriteLeftTop(Database[nb].GiveConversation());
              // write in the principal box

      Box1=n1;
      Box1G.WriteLeftTop(Database[n1].GiveConversation());
              // write in the box1
```

```
        if (Database[n1].GiveUtilisation()==0) Database[n1].ChangeUtilisation(-1);
                // Update the utilisation

        Box2=n2;
        Box2G.WriteLeftTop(Database[n2].GiveConversation());
                // write in the box2

        if (Database[n2].GiveUtilisation()==0) Database[n2].ChangeUtilisation(-1);
                // Update in the utilisation

        Box3=n3;
        Box3G.WriteLeftTop(Database[n3].GiveConversation());
                // write in the box3

        if (Database[n3].GiveUtilisation()==0) Database[n3].ChangeUtilisation(-1);
                // Update the utilisation

        Box4=n4;
        Box4G.WriteLeftTop(Database[n4].GiveConversation());
                // write in the box4

        if (Database[n4].GiveUtilisation()==0) Database[n4].ChangeUtilisation(-1);
                // Update the utilisation

// Update the shape mouse and the help line
        int xx, yy;
        Mouse.Event(xx,yy);
        int c=5;
        Which_box(xx,yy,c,Back_down, Back_up);
        Mouse.Show();

// Add a new element in the backtrack list.
        Backtrack.Add (nb, n1, n2, n3, n4);
}

////////////////////////////////////////////////////////////////////////////////

float Degree_Membership (st far * nn, int nouv)
// fuzzy algorithm.
// Compute the difference between 'nn' (pointer on the structure 'st')
// and 'nouv' (index in the database array).
//
// More the difference is little, more the return value will be high.

{
        int val, cb, gb, db, fb;


        static FDB far *tb;
        static FDB far *ti;


        float sol=0.0;
                // return value

        float degree;
```

```
                // temporary value

        fb = Database[nouv].GiveFrequency();
                // frequency of the 'nouv' item (nouv is an index on the database array)

// for each subject descriptor
        for (int i=0;i<Nb_subject;i++)
        {
        // cb is the center of the triangular fuzzy set of the subject[i] descriptor of the
        // item 'database[nouv]'
        // gb is the left limit of the triangular fuzzy set of the subject[i] descriptor of the
        // item 'database[nouv]'
        // db is the right limit of the triangular fuzzy set of the subject[i] descriptor of the
        // item 'database[nouv]'


                cb = Database[nouv].GiveSubject(i);

                // gb and db made according to the frequency.
                if (fb==1)
                {
                        gb = cb - largeur - 3;
                        db = cb + largeur -3;
                }
                if (fb==2)
                {
                        gb = cb - largeur - 1;
                        db = cb + largeur -1;
                }
                if (fb==3)
                {
                        gb = cb - largeur;
                        db = cb + largeur;
                }
                if (fb==4)
                {
                        gb = cb - largeur - 1;
                        db = cb + largeur + 1;
                }
                if (fb==5)
                {
                        gb = cb - largeur - 3;
                        db = cb + largeur + 3;
                }
                if (gb<-1) gb=-1;
                if (db>21) db=21;


                tb=new FDB;
                ti=new FDB;
                                // creation of the 2 object FDB.
                                // it is a creation of a empty fuzzy set.


                tb->FzyTriangleCurve(gb,cb,db);
                                // creation of the triangular fuzzy set of the subject[i] descriptor
```

```
                                // of the database[nouv]

        ti->FzyIntersection(&nn->psub[i]);
                                // make the intersection between ti (= fuzzy set empty) and
                                // nn.psub[i] (triangular fuzzy set of the subject[i] descriptor
                                // of the database[nb] - item in the principal box).

        ti->FzyIntersection(tb);
                                // make the intersection between ti and tb.

        ti->FzyDefuzzify(&degree);
                                // Find the highest value in ti.


        delete tb;
        delete ti;

// Modify 'degree' according as the weight of the descriptor
        float w=Subject_descriptor[i].GiveWeight();
                // w = weight of the subject[i] descriptor

                // more 'w' is high and more 'cb' is high, more degree will increase
        float p=w-1;
        p=p/10;
        int v;
        if (cb<11)
        {
                v=11-cb;
                degree=degree*(w-(v*p));
        }
        else
        {
                v=cb-10;
                degree=degree*(w+(v*p));
        }

        sol+=degree;
                                // Add this value at the return variable.
    }


// idem with the purpose descriptors.
    for (i=0;i<Nb_purpose;i++)
    {
        cb = Database[nouv].GivePurpose(i);
        if (fb==1)
        {
                gb = cb - largeur + 3;
                db = cb + largeur -3;
        }
        if (fb==2)
        {
                gb = cb - largeur + 1;
                db = cb + largeur -1;
        }
```

```
                if (fb==3)
                {
                        gb = cb - largeur;
                        db = cb + largeur;
                }
                if (fb==4)
                {
                        gb = cb - largeur - 1;
                        db = cb + largeur + 1;
                }
                if (fb==5)
                {
                        gb = cb - largeur - 3;
                        db = cb + largeur + 3;
                }
                if (gb<-1) gb=-1;
                if (db>21) db=21;

                tb = new FDB;
                ti = new FDB;

                tb->FzyTriangleCurve(gb,cb,db);
                ti->FzyIntersection(&nn->ppur[i]);
                ti->FzyIntersection(tb);
                ti->FzyDefuzzify(&degree);

                delete tb;
                delete ti;

                float w=Purpose_descriptor[i].GiveWeight();
                float p=w-1;
                p=p/10;
                int v;
                if (cb<11)
                {
                        v=11-cb;
                        degree=degree*(w-(v*p));
                }
                else
                {
                        v=cb-10;
                        degree=degree*(w+(v*p));
                }

                sol+=degree;
        }


// idem with the person descriptors.
        for (i=0;i<Nb_person;i++)
        {
                cb = Database[nouv].GivePerson(i);
                if (fb==1)
                {
                        gb = cb - largeur + 3;
```

```
                db = cb + largeur -3;
        }
        if (fb==2)
        {
                gb = cb - largeur + 1;
                db = cb + largeur -1;
        }
        if (fb==3)
        {
                gb = cb - largeur;
                db = cb + largeur;
        }
        if (fb==4)
        {
                gb = cb - largeur - 1;
                db = cb + largeur + 1;
        }
        if (fb==5)
        {
                gb = cb - largeur - 3;
                db = cb + largeur + 3;
        }
        if (gb<-1) gb=-1;
        if (db>21) db=21;

        tb = new FDB;
        ti = new FDB;

        tb->FzyTriangleCurve(gb,cb,db);
        ti->FzyIntersection(&nn->pper[i]);
        ti->FzyIntersection(tb);
        ti->FzyDefuzzify(&degree);

        delete tb;
        delete ti;

        float w=Person_descriptor[i].GiveWeight();
        float p=w-1;
        p=p/10;
        int v;
        if (cb<11)
        {
                v=11-cb;
                degree=degree*(w-(v*p));
        }
        else
        {
                v=cb-10;
                degree=degree*(w+(v*p));
        }

        sol+=degree;
    }
    return(sol);
}
```

```
#endif   // _SPECHBOX.CPP
```

A.3.3. Finish.cpp

```
////////////////////////////////////////////////////////////////////////////////
// -- Functions of the Finishing module.                                      //
////////////////////////////////////////////////////////////////////////////////
#ifndef _FINISH.HPP
#define _FINISH.HPP

#include <fstream.h>
#include <time.h>
#include <string.h>
#include <stdlib.h>


void Conversation_file(char intit[21],char * texte)
// Creation (or add) of the file 'conversa.out'.  This file will contains
// all conversation items speeched during the conversation .
//
{
        fstream file("conversa.out",ios::app);
                // pointer on the 'conversa.out' file

        time_t t;
        t=time(NULL);


        if (strcmp(intit,"")==0)
                file << "New conversation : " << ctime(&t);
                        // first element of the conversation

        else
        {
                time_t timer;
                struct tm *tblock;
                timer=time(NULL);
                tblock = localtime(&timer);

                        // Construction of the tp variable (=hour)
                char h[3];
                                // Hour

                char m[3];
                                // month

                char s[3];
                                // secondes

                char dp[2]=":";
                                // :

                char z[2]="0";
```

```
                                        // 0

                char tp[9];
                                        // time

                itoa(tblock->tm_hour,h,10);
                itoa(tblock->tm_min,m,10);
                itoa(tblock->tm_sec,s,10);
                                        // conversion in integer

                if (tblock->tm_hour<10)
                                        // if the hour < 10
                {
                        strcpy(tp,z);
                                        // add '0' before hour
                        strcat(tp,h);
                                        // add hour
                }
                else strcpy(tp,h);
                                        // if the hour > 9, add hour

                strcat(tp,dp);
                                        // add ':'

                if (tblock->tm_min<10) strcat(tp,z);
                                        // if min < 10, add '0'

                strcat(tp,m);
                                        // add minutes

                strcat(tp,dp);
                                        // add ':'

                if (tblock->tm_sec<10) strcat(tp,z);
                                        // if sec<10, add '0'

                strcat(tp,s);
                                        // add secondes


                file << "   " << tp << "  " << intit << texte << endl;
                        // write one line in the file (tp-intit-text)

                if (strcmp(intit,"EXIT BUTTON")==0) file << endl << endl;
                        // if it is the last element of the conversation
        }
}

////////////////////////////////////////////////////////////////////////////

void Ajustement_freq()
// Update of the frequency in the 'database.in' file.
//
// if conversation item had selected and used          -> +1
// if conversation item had selected but not used      -> -1
// if conversation item did not have selected or used -> no change
```

```
//
// This function rewrites the entire 'database.in' file.
//
{
        char bl[2];
        bl[0]=' ';
        bl[1]='\0';

        fstream file("database.in",ios::out|ios::trunc);
                // pointer on the 'database.in' file

// write the number of subject, purpose and person descriptors.
        file << Nb_subject << bl;
        file << Nb_purpose << bl;
        file << Nb_person << endl;

// write the names of the descriptors.
        for (int i=0;i<Nb_subject;i++)
                file << Subject_descriptor[i].GiveName() << bl;
        for (i=0;i<Nb_purpose;i++)
                file << Purpose_descriptor[i].GiveName() << bl;
        for (i=0;i<Nb_person;i++)
                file << Person_descriptor[i].GiveName() << bl;
        file << "frequency";

// for each conversation item.
        for (i=0;i<Nb_item;i++)
        {
                file <<endl;

        // write the value of the descriptors (subject, purpose and person)
                for (int j=0;j<Nb_subject;j++)
                        file << Database[i].GiveSubject(j) << bl;
                for (j=0;j<Nb_purpose;j++)
                        file << Database[i].GivePurpose(j) << bl;
                for (j=0;j<Nb_person;j++)
                        file << Database[i].GivePerson(j) << bl;

        // compute the new frequency.
                int frequency;
                frequency=((Database[i].GiveFrequency()) + Database[i].GiveUtilisation());
                if (frequency<1) frequency=1;
                if (frequency>5) frequency=5;

        // write the frequency.
                file << frequency;

        // write the conversation item.
                file << Database[i].GiveConversation()
        }

}

#endif   //_FINISH.HPP
```

A.3.4. Graphic.cpp

```
////////////////////////////////////////////////////////////////////////////
// -- Interface module.                                                    //
// -- All functions for the management of the interface                    //
////////////////////////////////////////////////////////////////////////////
#ifndef _GRAPHIC.HPP
#define _GRAPHIC.HPP

#include <graphics.h>
#include <iostream.h>
#include <constream.h>
#include <stdlib.h>


void InitGraph()
// Initialisation of the graphic mode
{
        int gdriver = DETECT, gmode, errorcode;

        initgraph(&gdriver, &gmode, "");
                // Initialises the graphics system

        errorcode = graphresult();
                // read result of initialisation

        if (errorcode != grOk)
                // an error occurred
        {
                cout << "\nGraphics error : " << grapherrormsg(errorcode);
                cout << "\nPress any key to halt...";
                getch();
                exit(1); // return with error code
        }


        installuserfont("litt.chr");
                // load a font file (litt.chr)

        installuserfont("trip.chr");
                // load a font file (trip.chr)

        installuserfont("tscr.chr");
                // load a font file (tscr.cnr)

        setbkcolor(1);
                // background color = blue

}

////////////////////////////////////////////////////////////////////////////

void Boites_normales()
// Draw the 5 boxes (Box principal and box1,2,3,4 on the screen
// with the option 'non-freezing'
```

```
{
        BoxPrincG.ChangeColor(14);
        Box1G.ChangeColor(14);
        Box2G.ChangeColor(14);
        Box3G.ChangeColor(14);
        Box4G.ChangeColor(14);
                // color = yellow

        BoxPrincG.Show();
        Box1G.Show();
        Box2G.Show();
        Box3G.Show();
        Box4G.Show();
                // show boxes on the screen

}

////////////////////////////////////////////////////////////////////////////

void Boites_freezes()
// Draw the 5 boxes (Box principal and box1,2,3,4 on the screen
// with the option 'freezing'
{
        BoxPrincG.ChangeColor(12);
        Box1G.ChangeColor(12);
        Box2G.ChangeColor(12);
        Box3G.ChangeColor(12);
        Box4G.ChangeColor(12);
                // color = red

        BoxPrincG.Show();
        Box1G.Show();
        Box2G.Show();
        Box3G.Show();
        Box4G.Show();
                // show boxes on the screen
}

////////////////////////////////////////////////////////////////////////////

void BackDownFuzzy()
// Draw '<' Button with green color
{
        BD.ChangeColor(2);
                // color = green

        BD.Show();
                // draw button

        setcolor(2);
                // color = green

        line(290,375,305,365);
        line(290,375,305,385);
                // draw '<' in the button
```

```
}

////////////////////////////////////////////////////////////////////////////

void BackDownNet()
// Draw '<' button with ligthgreen color
{
        BD.ChangeColor(10);
                // color = ligthgreen

        BD.Show();
                // draw button

        setcolor(10);
                // color = lightgreen

        line(290,375,305,365);
        line(290,375,305,385);
                // draw '<' in the button

}

////////////////////////////////////////////////////////////////////////////

void BackUpFuzzy()
// Draw '>' button with green color
{
        BU.ChangeColor(2);
                // color = green

        BU.Show();
                // draw button

        setcolor(2);
                // color = green

        line(420,375,405,385);
        line(420,375,405,365);
                // draw '>' in the button

}

////////////////////////////////////////////////////////////////////////////

void BackUpNet()
// Draw '>' button with green ligthcolor

{
        BU.ChangeColor(10);
                // color = lightgreen

        BU.Show();
                // draw button

        setcolor(10);
                // color = lightgreen
```

```
        line(420,375,405,385);
        line(420,375,405,365);
                // draw '>' in the button

}

////////////////////////////////////////////////////////////////////////////

void SetHelpLine()
// Draw help line
{
        setcolor(2);
                // color = green

        setlinestyle(0,1,1);
                // line = 1

        line(1,460,638,460);
                // Draw the line

        settextstyle(2, HORIZ_DIR,0);
                // set the current text characteristics

        setusercharsize(1,1,4,3);
                // set the size of the font

        outtextxy(20,461,"Left Mouse Button = ");
                // write on the screen

}

////////////////////////////////////////////////////////////////////////////

void HelpLine (int b)
// different options for the help line
{
        setcolor(2);
                // color = green

        settextstyle(2, HORIZ_DIR,0);
                // set the current text characteristics

        setusercharsize(1,1,4,3);
                // set the size of the font

        setviewport (130,461,500,475, 1);
                // set a new view port

        clearviewport();
                // clear the view port

        if (b==0) outtextxy(5,0,"Speak the sentence");
                // mouse in the Box Principal

        if (b==1) outtextxy(5,0,"Speak the sentence and search for other sentences");
```

```
                // mouse in the box 1,2,3,4

        if (b==5) outtextxy(5,0,"View more sentences");
                // mouse in the 'More' Button

        if (b==6) outtextxy(5,0,"To speak more than one sentence on the screen");
                // mouse in the 'Freeze/Unfreeze' button

        if (b==7) outtextxy(5,0,"Previous screen");
                // mouse in the '<' button

        if (b==8) outtextxy(5,0,"Next screen");
                // mouse in the '>' button

        if (b==9) outtextxy(5,0,"Help screen");
                // mouse in the 'help' button

        if (b==10) outtextxy(5,0,"To write a sentence");
                // mouse in the 'Unique text' box

        if (b==11) outtextxy(5,0,"Rapid sentences");
                // mouse in the Chat box

        if (b==12) outtextxy(5,0,"Exit button");
                // mouse in the 'exit' button

        if (b==14) outtextxy(5,0,"Type your text and press <return>");
                // When the user has click on the Unique text box


        setviewport(0,0,638,477,1);
                // set a new viewport

}

////////////////////////////////////////////////////////////////////////////

int Which_box(int x, int y, int & cursor, int bd, int bu)
// Return a integer according as (x,y) point, bd and bu.
//                (x,y) is the position of the mouse
//                bd =1 if the '<' button is active and =0 else
//                bu =1 if the '>' button is active and =0 else
// Update the shape pointer
// Update the helpline
{
        if (BoxPrincG.Inside(x,y))
                // (x,y) in Box Principal
        {
                if (cursor!=2)
                        // update cursor and the help line, if necessary
                {
                        Mouse.SetGCursor(MouthCursor);
                        HelpLine(0);
                        cursor=2;
                }
                return(0);
```

```
}

else if (Box1G.Inside(x,y))
        // (x,y) in Box 1
{
        if (cursor!=2)
                // update cursor and the help line, if necessary
        {
                Mouse.SetGCursor(MouthCursor);
                HelpLine(1);
                cursor=2;
        }
        return(1);
}

else if (Box2G.Inside(x,y))
        // (x,y) in Box 2
{
        if (cursor!=2)
                // update cursor and the help line, if necessary
        {
                Mouse.SetGCursor(MouthCursor);
                HelpLine(1);
                cursor=2;
        }
        return(2);
}

else if (Box3G.Inside(x,y))
        // (x,y) in Box 3
{
        if (cursor!=2)
                // update cursor and the help line, if necessary
        {
                Mouse.SetGCursor(MouthCursor);
                HelpLine(1);
                cursor=2;
        }
        return(3);
}

else if (Box4G.Inside(x,y))
        // (x,y) in Box 4
{
        if (cursor!=2)
                // update cursor and the help line, if necessary
        {
                Mouse.SetGCursor(MouthCursor);
                HelpLine(1);
                cursor=2;
        }
        return(4);
}

else if (More.Inside(x,y))
        // (x,y) in the 'More' Button
```

```
{
        if (cursor!=3)
                // update cursor and the help line, if necessary
        {
                Mouse.SetGCursor(HandCursor);
                HelpLine(5);
                cursor=3;
        }
        return(5);
}

else if (FreezeG.Inside(x,y))
        // (x,y) in the 'Freeze/unfreeze' Button
{
        if (cursor!=3)
                // update cursor and the help line, if necessary
        {
                Mouse.SetGCursor(HandCursor);
                HelpLine(6);
                cursor=3;
        }
        return(6);
}

else if (BD.Inside(x,y))
        // (x,y) in the '<' Button
{
        if (bd==1)
                // if '<' button is active
        {
                if (cursor!=3)
                        // update cursor and the help line, if necessary
                {
                        Mouse.SetGCursor(HandCursor);
                        HelpLine(7);
                        cursor=3;
                }
                return 7;
        }

        else
                // if '<' button is not active
        {
                if (cursor!=1)
                        // update cursor and the help line, if necessary
        {
                        Mouse.SetGCursor(CrossCursor);
                        HelpLine(13);
                        cursor=1;
                }
                return (13);
        }
}

else if (BU.Inside(x,y))
        // (x,y) in the '>' Button
```

```
{
        if (bu==1)
                // if the '>' button is active
        {
                if (cursor!=3)
                        // update cursor and the help line, if necessary
                {
                        Mouse.SetGCursor(HandCursor);
                        HelpLine(8);
                        cursor=3;
                }
                return (8);
        }

        else
                // if the '>' button is not active
        {
                if (cursor!=1)
                        // update cursor and the help line, if necessary
                {
                        Mouse.SetGCursor(CrossCursor);
                        HelpLine(13);
                        cursor=1;
                }
                return (13);
        }
}

else if (Help.Inside(x,y))
        // (x,y) in the 'Help' Button
{
        if (cursor!=3)
                // update cursor and the help line, if necessary
        {
                Mouse.SetGCursor(HandCursor);
                HelpLine(9);
                cursor=3;
        }
        return (9);
}

else if (Text.Inside(x,y))
        // (x,y) in the 'Unique Text' box
{
        if (cursor!=4)
                // update cursor and the help line, if necessary
        {
                Mouse.SetGCursor(IBeamCursor);
                HelpLine(10);
                cursor=4;
        }
        return(10);
}

else if (ChBox.Inside(x,y))
         // (x,y) in the Chat Box
```

```
        {
                if (cursor!=2)
                        // update cursor and the help line, if necessary
                {
                        Mouse.SetGCursor(MouthCursor);
                        HelpLine(11);
                        cursor=2;
                }
                return(11);
        }

        else if (Exit.Inside(x,y))
                // (x,y) in the 'Exit' Button
        {
                if (cursor!=3)
                        // update cursor and the help line, if necessary
                {
                        Mouse.SetGCursor(HandCursor);
                        HelpLine(12);
                        cursor=3;
                }
                return(12);
        }

        else
                // Default
        {
                if (cursor!=1)
                        // update cursor and the help line, if necessary
                {
                        Mouse.SetGCursor(CrossCursor);
                        HelpLine(13);
                        cursor=1;
                }
                return (13);
        }
}


#endif   // _GRAPHIC.CPP
```

A.3.5. Textchat.cpp

```
////////////////////////////////////////////////////////////////////////////////
// -- Unique Text and Chatbox module                                          //
////////////////////////////////////////////////////////////////////////////////
#ifndef _TEXTCHAT.CPP
#define  _TEXTCHAT.CPP

#include <graphics.h>
#include <constream.h>
#include <dos.h>

#define Text_Max 64
```

```
void TextBox()
// Management of the Unique text
{
        HelpLine(14);
                // change the help line

        char text[Text_Max+1];
        text[0]='\0';

        char c;
                // c for reading letter typed by user

        letter[2];
                // letter[] for writing the letter typed to the screen

        cursor[2];
        cursor[0]='_';
        cursor[1]='\0';
                // cursor[] for displaying the text cursor in unique box

        int  Ux = 10, Uy = 5,       // text coords for displaying text to the screen
        n = 0;                      // counter - char array element access & recording
                                    // whether any keys pressed - for backspacing


// prepare for writing to screen
        setviewport(21, 421, 619, 449, 1);
        clearviewport();
        setviewport(20, 420, 620, 450, 1);
        setusercharsize(3,2,3,2);
        settextstyle(2, HORIZ_DIR, 0);

// show cursor
        setcolor(14);
        outtextxy(Ux, Uy, cursor);

// get key from user
        c = getch();

// while not <return> to end & say string
        while (c != 13)
        {
                if (c == 8)     // if backspace pressed
                {
                        if (n > 0)          // only if chars already in "text" string
                        {                   // "deletes" character from screen by writing over it
                                            // in white and writing over it in unique_text char array
                                letter[0] = text[n-1];// get letter to be erased
                                letter[1] = '\0';
                                Ux -= textwidth(letter);          // go to coords of the char
                                setcolor(1);
                                outtextxy(Ux, Uy, letter);        // write over it (in white)
                                                                  // write over cursor in white
                                outtextxy(Ux+textwidth(letter),Uy,cursor);
```

```
                        setcolor(14);
                        outtextxy(Ux, Uy, cursor);   // & write it in previous space
                        n--;                  // "erase" letter from char array
                        text[n] = '\0';   // by writing over with NULL (end
                                                        // of string) character.
                }
                else
                {    // if no characters in string beep because nothing
                        // there to delete
                        sound(900);
                        delay(100);
                        nosound();
                };

        }
        else   // record key pressed and display it in the text box
        {
                if (n>Text_Max)
                        // if the cursor is in the end of the text box
                {
                        sound(900);
                        delay(100);
                        nosound();
                }
                else
                {
                        text[n] = c;
                        text[n+1] = '\0';
                                // Update the text variable

                        n++;
                        letter[0] = c;
                        letter[1] = '\0';  // Null terminating char of string

                // move cursor along one space
                        setcolor(1);
                        outtextxy(Ux,Uy, cursor);
                        setcolor(14);
                        outtextxy(Ux+textwidth(letter),Uy,cursor);
                        outtextxy(Ux,Uy,letter);
                        Ux += textwidth(letter);
                }
        }
        c = getch();        // get next text char from user
    }

        // remove cursor from line
setcolor(1);
outtextxy(Ux,Uy,cursor);

        // speak the text via the synthesiser
apollo(text,1);

        // Update the 'Conversa.out' file
Conversation_file("UNIQUE TEXT : ",text);
```

```
                // new view port
    setviewport(0,0,638,477,1);

                // Update the shape pointer and the help line
    int xx, yy;
    Mouse.Event(xx,yy);
    int cuu=5;
    Which_box(xx,yy,cuu,Back_down, Back_up);

}

//////////////////////////////////////////////////////////////////////////////

void Chat_draw()
// Draw the Chat box

{
        ChBox.Show();
                // main rectangle

        setcolor (10);
        setlinestyle(0,1,3);
        line (575,20,575,290);
                // set color ant thickness of lines

        line (530,53,620,53);
        line (530,87,620,87);
        line (530,121,620,121);
        line (530,155,620,155);
        line (530,189,620,189);
        line (530,223,620,223);
        line (530,257,620,257);
                // Draw lines

        settextstyle(2,0,0);
        setusercharsize(1,1,1,1);

        outtextxy(540,30,"Hello");
        outtextxy(587,30,"Fine");

        outtextxy(533,63,"Wrap-up");
        outtextxy(590,63,"Bye");

        outtextxy(545,97,"Yes");
        outtextxy(592,97,"No");

        outtextxy(540,131,"Agree");
        outtextxy(590,126,"Not");
        outtextxy(585,136,"agree");

        outtextxy(540,165,"Good");
        outtextxy(590,165,"Bad");

        outtextxy(540,194,"Don't");
        outtextxy(542,204,"know");
        outtextxy(580,199,"Thanks");
```

```
        outtextxy(542,233,"Wait");
        outtextxy(587,233,"More");

        outtextxy(537,267,"Sorry");
        outtextxy(580,267,"Uh-huh");
                // Write text

}

////////////////////////////////////////////////////////////////////////////

void Chat_speak(int px, int py)
// speak via the synthesiser the Chatbox item according as (px,py)
{
        if (px<576)
        {
                if ((py>20)  && (py<53))  apollo ("Hello, How are you?",1);
                if ((py>52)  && (py<87))  apollo ("Well, that's about it",1);
                if ((py>86)  && (py<121)) apollo ("Yes",1);
                if ((py>120) && (py<155)) apollo ("I agree",1);
                if ((py>154) && (py<189)) apollo ("That's good",1);
                if ((py>188) && (py<223)) apollo ("I don't know",1);
                if ((py>222) && (py<257)) apollo ("Please, wait while I find the words",1);
                if ((py>256) && (py<290)) apollo ("Sorry",1);
        }

        else
        {
                if ((py>20)  && (py<53))  apollo ("I'm fine, thank you",1);
                if ((py>52)  && (py<87))  apollo ("Good bye",1);
                if ((py>86)  && (py<121)) apollo ("No",1);
                if ((py>120) && (py<155)) apollo ("I disagree",1);
                if ((py>154) && (py<189)) apollo ("That's bad",1);
                if ((py>188) && (py<223)) apollo ("Thank you",1);
                if ((py>222) && (py<257)) apollo ("Please, tell me more about it",1);
                if ((py>256) && (py<290)) apollo ("Uh-huh",1);
        }

        Conversation_file("CHAT BOX","");
                // Udate the 'Conversa.out' file
}


#endif   // _TEXTCHAT.CPP
```

A.3.6. Help.cpp

```
////////////////////////////////////////////////////////////////////////////
// -- Help module                                                         //
////////////////////////////////////////////////////////////////////////////
#ifndef _HELP.CPP
#define _HELP.CPP
```

```
#include <graphics.h>


// Global variables for the help screen
Button Ok("OK",10,3,7,573,440,45,20);
        // 'Ok' button
Button HP("",2,3,7,353,440,45,20);
        // '<' button
Button HN("",10,3,7,463,440,45,20);
        // '>' button


////////////////////////////////////////////////////////////////////////////////

void HelpPreviousFuzzy()
// Draw '<' button with green color
{
        HP.ChangeColor(2);
                // color of the button = green

        HP.Show();
                // show button

        setcolor(2);
                // color =green

        line(345,440,360,430);
        line(345,440,360,450);
                // draw '<'

}

////////////////////////////////////////////////////////////////////////////////

void HelpPreviousNet()
// Draw '<' button with ligth green color

{
        HP.ChangeColor(10);
                // color of the button = ligth green

        HP.Show();
                // show button

        setcolor(10);
                // color = light green

        line(345,440,360,430);
        line(345,440,360,450);
                // draw '<'

}

////////////////////////////////////////////////////////////////////////////////

void HelpNextFuzzy()
```

```
// Draw '>' button with green color
{
        HN.ChangeColor(2);
                // color of the button = green

        HN.Show();
                // show button

        setcolor(2);
                // color =green

        line(475,440,460,450);
        line(475,440,460,430);
                // draw '>'

}

////////////////////////////////////////////////////////////////////////////

void HelpNextNet()
// Draw '>' button with light green color
{
        HN.ChangeColor(10);
                // color of the button = light green

        HN.Show();
                // show button

        setcolor(10);
                // color = light green

        line(475,440,460,450);
        line(475,440,460,430);
                // draw '>'

}

////////////////////////////////////////////////////////////////////////////

void MapMin()
// draw the FuzzyChat interface reduced
{
        setlinestyle(0,1,1);
                // set the line style

        setcolor(14);
                // color = yellow

        rectangle(400,40,613,201);
                // the screen
        rectangle(406,46,563,70);
                // box princ
        rectangle(406,77,543,88);
                // box 1
        rectangle(413,97,550,108);
                // box 2
```

```
        rectangle(420,117,557,128);
                // box 3
        rectangle(427,137,563,148);
                // box 4
        rectangle(406,180,607,190);
                // unique text

        setcolor(2);
                // color = green

        rectangle(577,46,607,137);
                // chatbox
        ellipse(427,165,0,360,15,7);
                // more button
        ellipse(463,165,0,360,15,7);
                // freeze/unfreeze button
        ellipse(500,165,0,360,15,7);
                // < button
        ellipse(537,165,0,360,15,7);
                // > button
        ellipse(591,165,0,360,15,7);
                // help button

        line(401,193,612,193);
                // helpline
        rectangle(577,194,607,200);
                // exit button

// write numbers in buttons and boxes
        settextstyle(2, HORIZ_DIR,0);
        setusercharsize(1,1,1,1);
        setcolor(12);
        outtextxy(484,52,"1");
        outtextxy(474,76,"2");
        outtextxy(481,96,"3");
        outtextxy(488,116,"4");
        outtextxy(495,136,"5");
        outtextxy(590,84,"6");
        outtextxy(425,158,"7");
        outtextxy(461,158,"8");
        outtextxy(498,158,"9");
        outtextxy(532,158,"10");
        outtextxy(586,158,"11");
        outtextxy(501,179,"12");
        outtextxy(506,205,"13");
        outtextxy(600,205,"14");
        line(497,211,504,211);
        line(490,197,497,211);
        line(591,211,598,211);
        line(584,197,591,211);
}

////////////////////////////////////////////////////////////////////////////

void Title()
// write on the screen the 'help' title
```

```
{
        setcolor(10);
        settextstyle(7,0,0);
        setusercharsize(2,3,1,2);
        outtextxy(245,15,"Help Screen");
        setlinestyle(0,0,1);
        line (240,35,380,35);
        line (240,40,380,40);
}

////////////////////////////////////////////////////////////////////////////////

void Signature()
// write the name of software and his designer
{
        Rectangle Sign(10,1,1,20,420,170,460,0);
        Sign.Show();
        outtextxy(30,425,"FUZZYCHAT 1.0, 1994");
        outtextxy(30,440,"by Marc Uytdenbroek.");

}

////////////////////////////////////////////////////////////////////////////////

void HelpTextFirst()
// write the first help screen
{
        Mouse.Hide();
                // hide the mouse

        cleardevice();
                // clear device

        Title();
                // write the help title

        MapMin();
                // draw the map of the fuzzychat interface

        Ok.ChangeColor(10);
        Ok.Show();
        HelpPreviousFuzzy();
        HelpNextNet();
                // update of < and > buttons

        setcolor(14);
        settextstyle(2,0,0);
        setusercharsize(6,5,4,3);
        outtextxy(20,55,"FUZZYCHAT is an augmentative communication system for");
        outtextxy(20,70,"non-verbal severely physically impaired people.");

        outtextxy(40,122,"This box contains the text item just spoken via");
        outtextxy(40,137,"the speech synthesiser (if the 'Freeze' button is");
        outtextxy(40,152,"is not selected). If you click on this box, the");
        outtextxy(40,167,"text item will be repeated.");
```

```
        outtextxy(100,189,"These boxes contain the four text");
        outtextxy(40,204,"predictions closest to (1).");
        outtextxy(40,223,"- If the 'Freeze' button is not selected, the");
        outtextxy(40,238,"text item in the selected box will be spoken, via the speech synthesiser,
when you");
        outtextxy(40,253,"click on one of these boxes. This text item will then go into box (1) and
four");
        outtextxy(40,268,"other text items will be found and placed the other four boxes. These text
items will");
        outtextxy(40,283,"be the closest predictions to the text in box (1).");
        outtextxy(40,302,"- If the 'Freeze' button is selected, the text item in the selected box will
be spoken,");
        outtextxy(40,317,"via the speech synthesiser when you click on one of these boxes.");

        outtextxy(40,339,"This box contains buttons that allow you to say (via the speech synthesiser)
quick-fire");
        outtextxy(40,354,"phrases.");

        outtextxy(40,376,"The 'More' button causes a new search to find the next four text items
closest to the");
        outtextxy(40,391,"text in box (1). The boxes (2), (3), (4) and (5) will be updated.");

        setcolor(12);
        outtextxy(20,100,"Description of left mouse button functions:");
        outtextxy(20,122,"(1)");
        outtextxy(20,189,"(2)(3)(4)(5)");
        outtextxy(20,339,"(6)");
        outtextxy(20,376,"(7)");

        Signature();
                // write the name of the software and his designer

        Mouse.Show();
                // show the mouse
}

////////////////////////////////////////////////////////////////////////////////

void HelpTextSecond()
// write the second help screen
{
        Mouse.Hide();
                // hide the mouse

        cleardevice();
                // clear device

        Title();
                // write the help title

        MapMin();
                // draw the map of the fuzzychat interface

        Ok.ChangeColor(10);
        Ok.Show();
        HelpPreviousNet();
```

```
        HelpNextFuzzy();

        setcolor(14);
        settextstyle(2,0,0);
        setusercharsize(6,5,4,3);

        outtextxy(40,55,"- The 'Freeze' button allows the text items in boxes");
        outtextxy(40,70,"(1), (2), (3), (4) and (5) to be frozen.  This means");
        outtextxy(40,85,"that when you click on one of these boxes, there is");
        outtextxy(40,100,"not a new search for the closest item.");
        outtextxy(40,119,"- The 'Unfreeze' button allows the text items in");
        outtextxy(40,134,"boxes (1), (2), (3), (4) and (5) to be unfrozen. This");
        outtextxy(40,149,"means that when you click on one of these boxes,");
        outtextxy(40,164,"there is a new search for the closest item.");

        outtextxy(40,186,"The '<' button allows you to come back to the");
        outtextxy(40,201,"previous screen, if a previous screen exists.");

        outtextxy(40,223,"The '>' button allows you to go to the next");
        outtextxy(40,238,"screen, if a next screen exists.");

        outtextxy(40,260,"The 'Help' button displays this screen.");

        outtextxy(40,282,"This box allows you to type any text. After you have typed the text, push
<return>.");
        outtextxy(40,297,"The text will be then spoken via the speech synthesiser.");

        outtextxy(40,319,"This is a help line.");

        outtextxy(40,341,"Clicking on this button will terminate a FUZZYCHAT session.");

        setcolor(12);
        outtextxy(20,55,"(8)");
        outtextxy(20,186,"(9)");
        outtextxy(13,223,"(10)");
        outtextxy(13,260,"(11)");
        outtextxy(13,282,"(12)");
        outtextxy(13,319,"(13)");
        outtextxy(13,341,"(14)");

        Signature();
                // write the name of the software and his designer

        Mouse.Show();
                // show the mouse
}

////////////////////////////////////////////////////////////////////////////////

void help(int f, int bd, int bu)
// main function of the help module - Management of the mouse
{
        Conversation_file("HELP BUTTON (B)","");
                // Update the 'conversa.out' file

        HelpTextFirst();
```

```
        // display the first help screen

int hp=0;
int hn=1;

int c=1;
        // Cursors : 1=Cross, 2=Hand;

Mouse.SetGCursor(CrossCursor);
        // cursor = cross

int ok=0;
unsigned e;
int mx, my;
        // local variable for the management of the mouse

do
        // main loop
{
        e=Mouse.Event(mx, my);
                // Waiting for a mouse event : (mx,my) = position of the mouse.

        if (Ok.Inside(mx,my))
                // if (mx,my) is inside of the 'Ok' button
        {
                if (c!=2)
                        // if the shape pointer is not the 'Hand'
                {
                        Mouse.SetGCursor(HandCursor);
                        c=2;
                                // shape pointer = 'Hand'
                }
                if (e==LMouseDown)
                        // left button (of the mouse) pushed
                {
                        Mouse.Hide();
                        Ok.ChangeColor(2);
                        Ok.Show();
                        Mouse.Show();
                        Mouse.WaitForEvent(LMouseUp,mx,my);
                        ok=1;    // end of help screens = true
                }
        }

        else if (HP.Inside(mx,my))
                // if (mx,my) is inside the '<' button
        {
                if (hp==1)
                        // if the '<' button is active
                {
                        if (c!=2)
                                // if the shape pointer is not the 'Hand'
                        {
                                Mouse.SetGCursor(HandCursor);
                                c=2;
                                        // shape pointer = 'Hand'
```

```
                }
                if (e==LMouseDown)
                        // if left button (of the mouse) pushed
                {
                        Mouse.Hide();
                        HelpPreviousFuzzy();
                        Mouse.Show();
                        Mouse.WaitForEvent(LMouseUp,mx,my);
                        hp=0;hn=1;
                        HelpTextFirst();
                                // first help screen
                }
        }
        else if (c!=1)
                // if the '<' button is not active and if the shape pointer
                // is not a 'Cross'
        {
                Mouse.SetGCursor(CrossCursor);
                c=1;
                        // shape pointer = 'Cross'
        }
}

else if (HN.Inside(mx,my))
        // if (mx,my) is inside the '>' button
{
        if (hn==1)
                // if the '>' button is active
        {
                if (c!=2)
                        // if the shape pointer is not the 'Hand'
                {
                        Mouse.SetGCursor(HandCursor);
                        c=2;
                                // shape pointer = 'Hand'
                }
                if (e==LMouseDown)
                        // if left button (of the mouse) pushed
                {
                        Mouse.Hide();
                        HelpNextFuzzy();
                        Mouse.Show();
                        Mouse.WaitForEvent(LMouseUp,mx,my);
                        hn=0;hp=1;
                        HelpTextSecond();
                                // second help screen
                }
        }
        else if (c!=1)
                // if the '>' button is not active and if the shape pointer
                // is not a 'Cross'
        {
                Mouse.SetGCursor(CrossCursor);
                c=1;
                        // shape pointer = 'Cross'
        }
```

```
                }

                else if (c!=1)
                        // if (mx,my) is not inside '<' '>' and 'or' buttons and
                        // if shape pointer is not the 'Cross'
                {
                        Mouse.SetGCursor(CrossCursor);
                        c=1;
                                // shape pointer ='Cross
                }
        }
        while (!ok);
                // while not ok

        Mouse.Hide();
        cleardevice();


// Resauration of the main fuzzy screen
        if (f==1)
                // if screen was 'freeze'
        {
                Boites_freezes();
                        // draw the 5 boxes
                Unfreeze.Show();
                        // draw the 'unfreeze' button
        }
        else
                // if screen was 'unfreeze'
        {
                Boites_normales();
                        // draw the 5 boxes
                FreezeG.Show();
                        // draw the 'freeze' button
        }

        if (bd==1) BackDownNet();
                // if the '<' button was active
        else BackDownFuzzy();
                // if the '<' button wasn't active

        if (bu==1) BackUpNet();
                // if the '>' button was active
        else BackUpFuzzy();
                // if the '<' button wasn't active

        Conversation_file("HELP BUTTON (E)","");
                // update the 'conversa.out' file

        More.Show();
                // draw the 'more' button

        Help.Show();
                // draw the 'help' button

        Text.Show();
```

```
                // draw the 'unique text' box

        Exit.Show();
        Exit.WriteCenter("E  x  i  t");
                // draw the 'exit' button

        SetHelpLine();
                // draw the help line

        Chat_draw();
                // draw the chatbox

        BoxPrincG.WriteLeftTop(Database[BoxPrinc].GiveConversation());
                // draw the text item (who was in before) in the principal box

        Box1G.WriteLeftTop(Database[Box1].GiveConversation());
                // draw the text item (who was in before) in the box 1
        Box2G.WriteLeftTop(Database[Box2].GiveConversation());
                // draw the text item (who was in before) in the box 2
        Box3G.WriteLeftTop(Database[Box3].GiveConversation());
                // draw the text item (who was in before) in the box 3
        Box4G.WriteLeftTop(Database[Box4].GiveConversation());
                // draw the text item (who was in before) in the box 4


        Mouse.Show();
}


#endif   // _HELP.CPP
```

A.3.7. Synth.cpp

```
////////////////////////////////////////////////////////////////////////////////
// This routine sends the string pointed to by 'p' to a Dolphin               //
// Apollo text-to-speech synthesiser connected to the serial port             //
// (com) of the computer.                                                     //
////////////////////////////////////////////////////////////////////////////////
#ifndef _SYNTH.CPP
#define _SYNTH.CPP

#include <dos.h>


#define SERIAL 0x14
        // ROM BIOS interrupt for serial port I/O


void apollo(char *p, int com)
{
        if (Synthesiser==1)
                        // if Synthesiser is ON

        {
                union REGS r;
```

```
// Initialise serial port

                r.h.ah = 0;
                                // initialise
                r.h.al = 227;
                                // baud rate 9600; no parity, 8 data bits, 1 stop bit
                r.x.dx =com;
                                // use COM1 if com=0, use COM2 if com=1
                int86(SERIAL, &r, &r);
                                // general 8086 software interrupt


// While character is not end of string, send to synthesiser

                while (*p != '\0')
                {
                        r.h.ah = 1;
                                // write
                        r.h.al = *p;
                                // assign ASCII value of character to AL
                        r.x.dx = com;
                                // use COM1 if com=0, use COM2 if com=1
                        int86(SERIAL, &r, &r);
                                // general 8086 software interrupt
                        p++;
                                // get next character
                }


// Flush: send a carriage return to synthesiser

                r.h.ah = 1;
                        // write
                r.h.al = 13;
                        // assign a carriage return to AL
                r.x.dx = com;
                        // use COM1 if com=0, use COM2 if com=1
                int86(SERIAL, &r, &r);
                        // general 8086 software interrupt

        }

        return;
}


#endif   // _SYNTH.CPP
```